Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Diseño y caracterización de un sonar de seguimiento basado en Arduino

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación

AUTOR: Artur Romero Hazas

DIRECTOR: José María González Arbesú
CODIRECTOR: Marta González Rodríguez

FECHA: 22 de octubre del 2019

Título: Diseño y caracterización de un sonar de seguimiento basado en Arduino

Autor: Artur Romero Hazas

Director: José María González Arbesú
Codirector: Marta González Rodríguez

Fecha: 22 de octubre del 2019

Resumen

Este documento contiene toda la información necesaria para realizar la fabricación y caracterización de un sonar de seguimiento basado en Arduino. El trabajo tiene un fin educativo y pretende proporcionar los conocimientos necesarios para desarrollar el sistema de seguimiento por cuenta propia. La presente memoria incluye la base teórica, matemática, electrónica, mecánica e informática requerida para el desarrollo del sonar de seguimiento.

Este sistema sonar de seguimiento se basa en el uso sensores de ultrasonidos, para localizar un objeto, posicionarlo en un sistema de coordenadas y rastrearlo mediante un sistema mecánico dotado de motores, integrados en un entorno de software y hardware de Arduino.

Los sistemas sonar de seguimiento pueden definirse en dos dimensiones, considerando únicamente dos coordenadas variables (por ejemplo: *x* e *y*) y por tanto con un único eje de rotación, en azimut; o en tres dimensiones, considerando una tercera variable (que podría ser la *z)*, y añadiendo un segundo eje de rotación, en elevación.

Empezaremos analizando las aplicaciones de los sistemas de posicionamiento y de seguimiento actuales y los equipos existentes, la base electrónica que engloba el proyecto, la teoría matemática y algoritmos para los cálculos de posición y triangulación de objetivos y el desarrollo del software completo.

Continuaremos con el diseño completo de un sonar en dos dimensiones (2D), como paso previo para el diseño de un sonar en tres dimensiones (3D) y sus posibles aplicaciones. Para el diseño del sistema mecánico se utilizará software CAD, en este caso SOLIDWORKS y se realizará la impresión 3D del mismo.

El ensamblaje englobará todos los componentes, tanto mecánicos como electrónicos, definiendo el sistema de seguimiento por completo, y listando todos las piezas y materiales utilizados, sus costes relacionados y el presupuesto final; incluyendo los planos de las piezas en el apéndice.

Finalmente, analizaremos el funcionamiento del sistema, los resultados teóricos y prácticos obtenidos, estudiaremos las limitaciones del sistema y las posibles mejoras futuras.

Title: Diseño y caracterización de un sonar de seguimiento basado en Arduino

Author: Artur Romero Hazas

Director: José María González Arbesú
Codirector: Marta González Rodríguez

Date: October 22nd, 2019

Overview

In this document you will find all the required information to carry out the design and characterization of a tracking sonar based on Arduino. The project has an educational purpose and aims to provide the necessary knowledge to develop the full tracking sonar system on its own. This report includes the theoretical, mathematical, electronic, mechanical and software basis to realize the complete project.

This sonar tracking system is based on the use of ultrasonic sensors, to locate an object, position it in a coordinate system and track it using a mechanical mechanism and motors, integrated into an Arduino software and hardware environment.

The sonar tracking systems can be defined in two dimensions, considering only two variable coordinates (for example: x and y) and therefore with a single axis of rotation, in azimuth; or in three dimensions, considering a third variable that could be the z, and adding a second axis of rotation, the elevation one.

We will begin by analyzing the existing systems and equipment related to a sonar, the electronic basis that encompasses the project, the mathematical theory and algorithms for the calculations of position and triangulation of objectives and the development of the complete code and software.

We will continue with the complete design of a sonar in two dimensions (2D), the design of a sonar in three dimensions (3D) and its possible applications. CAD software will be used for the design of the mechanical system, in this case SOLIDWORKS and 3D printing will be performed.

The assembly will include all components, both mechanical and electronic, defining the monitoring system completely, and listing all the parts and materials used, their related costs and the final budget; including all the schematics in the appendix.

Finally, we will analyze the operation of the system, the theoretical and practical results obtained, we will study the limitations of the system and possible future improvements or next steps.

Dedicatoria

Quiero agradecer todo el guiado y el trabajo realizado por mis directores durante todo el proyecto. Dedicarle este trabajo a la familia y a mi pareja por todo el amor, esfuerzo y apoyo diario.

ÍNDEX

CAPÍTULO 1. INTRODUCCIÓN

1.1. Objetivos del proyecto

El objetivo de este documento es detallar toda la información necesaria para poder desarrollar y caracterizar un sonar de seguimiento con hardware y software basado en Arduino [1] con fines educativos.

El requisito fundamental de este proyecto es mantener un carácter educativo, aplicable a nivel universitario, para poder desarrollar y estudiar el sonar de seguimiento en asignaturas relacionadas con la radiolocalización. Por ejemplo, en la asignatura de Radiolocalización de los estudios de Grado en Ingeniería de Sistemas de Telecomunicación y Grado en Ingeniería de Sistemas Aeroespaciales que se imparten en la EETAC de la UPC. El documento incluye toda la base teórica necesaria relacionada con electrónica, matemáticas, programación e informática; con el objetivo de complementar y refrescar la base de conocimiento del alumnado para poder implementar el sistema por completo.

Por otro lado, también se listan y se estudian todos los componentes utilizados, el diseño de los modelos y el ensamblaje completo del sistema.

El desarrollo del sistema sonar de seguimiento se basará en software y hardware de Arduino, englobando los sensores de ultrasonidos y los motores a usar, la placa microcontroladora Arduino para obtener los datos de los sensores y actuar sobre los motores desde un programa en el ordenador y el desarrollo de código y software para controlar el sistema.

Finalmente, en el proyecto se analizarán los resultados obtenidos, estimar el coste total, encontrar las limitaciones del sistema y plantear posibles futuras mejoras.

Todos los recursos relacionados con el proyecto están disponibles en el siguiente enlace: *http://bit.ly/recursos_TFG_sonar* .

1.2. Especificaciones del proyecto

El software necesario para desarrollar el proyecto ha ser de uso libre o disponible para estudiantes universitarios. Todos los componentes requeridos estarán listados y referenciados para poder ser obtenidos, sin olvidar el carácter didáctico de los mismos y usando el mínimo número posible de elementos con el objetivo de que el sistema final resulte lo más simple y económico posible.

Desde un principio, el proyecto debe ser desarrollado en el entorno Arduino. El sistema de seguimiento se basará en el sensor de ultrasonidos HC-SR04 para leer las distancias y el motor 28BYJ-48 para realizar los movimientos necesarios.

1.3. Antecedentes

De los posibles sistemas y equipos de posicionamiento y localización existentes, este trabajo se centrará en aquellos que usan ondas sonoras como medio de detección, el sonar.

El sonar (derivado del inglés: *sound navigation and ranging*) es un sistema y técnica utilizada como medio de localización acústica similar al radar, que utiliza la propagación del sonido, mayoritariamente en aplicaciones bajo agua, con el objetivo de detectar objetos, comunicarse y navegar.

El uso del primer sonar fue registrado en 1490 por Leonardo da Vinci, que usaba un artilugio formado por un tubo que se sumergía en el agua para escuchar y detectar la presencia de barcos. Aunque no es hasta 1912 que se impulsa el uso de la ecolocalización, protagonista del desastre de Titanic. La primera patente sobre un dispositivo sonar fue de un meteorólogo inglés, Lewis Richardson, en el mismo año. La Primera Guerra Mundial y la Segunda Guerra Mundial instigó las investigaciones sobre el uso del sonido para detectar submarinos, empleando micrófonos subacuáticos o hidrófonos.

Los equipos sonar emiten ondas sonoras, generalmente no audibles, a una frecuencia superior a 20 kHz, en la banda de ultrasonido.

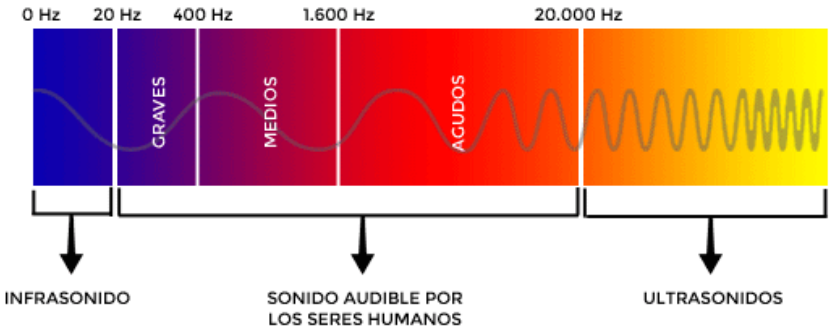

Fig. 1.1. Espectro sonoro de 0 Hz a 20 kHz [2]

Los equipos ultrasónicos también son utilizados en aplicaciones aéreas, mediante instrumentos llamados sodar (del inglés: *sonic detection and ranging*). El sodar se emplea con fines meteorológicos para medir y estudiar los vientos de la capa baja de la atmósfera.

La eficacia y el rendimiento de detección, localización y clasificación de objetos de un sonar depende de muchos factores como: el equipo emisor y receptor, el entorno o medio de propagación y de la reflexión sonora del objetivo. Con todo ello, los equipos sonar son muy sensibles a factores externos y dependientes de la velocidad de propagación del sonido que varía según el medio usado y las condiciones ambientales. Además de considerar todas las fuentes posibles de

ruido y reverberación sonora producida por otros objetos que generarían interferencias en el equipo receptor.

Centrándonos en un sonar activo monoestático, donde emisor y receptor se encuentran en el mismo lugar, el emisor genera un pulso de sonido que se transfiere al medio y el receptor recibe el eco provocado por cierto objeto. Para obtener la distancia al objetivo, se mide el tiempo transcurrido desde la emisión del pulso a la recepción del eco y se calcula la longitud viajada a partir de la velocidad de propagación del sonido en el medio de operación.

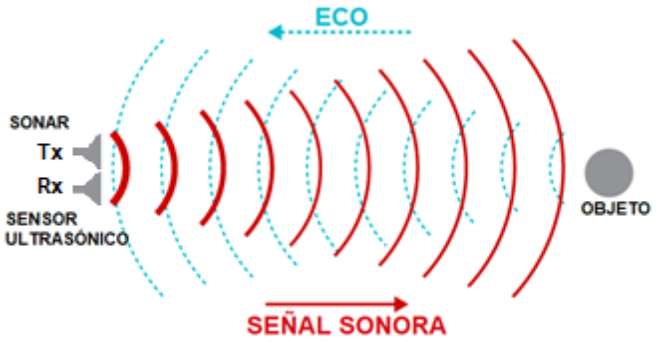

Fig. 1.2. Escenario de detección mediante sonar

Para aplicaciones de localización, se utilizan varios emisores/receptores con posición conocida para medir el tiempo de llegada relativa en cada uno de ellos y poder posicionar el objetivo.

A continuación, se listan las principales aplicaciones de equipos sonar:

- Aplicaciones civiles:
  - Pesca: para detección de barco y bancos de peces.
  - Naval: para localización de barcos, detección de objetos y cálculo de profundidad y velocidad del buque.
  - ROVs (Remote Operated Vehicle): vehículos submarinos no tripulados para búsqueda, rescate e investigación submarina.
  - Aeronáutica: actualmente utilizados como boyas o sistema de localización de emergencia en caso de accidente en el mar.
- Aplicaciones militares:
  - Sonar submarino: permite la localización y detección de barcos, submarinos y obstáculos; además de comunicaciones subacuáticas, vigilancia y seguridad marina,
  - Torpedo y mina sonar: para localizar y perseguir un blanco.
  - Sonar aéreo: utilizado en aviones y helicópteros para rastrear barcos y submarinos.
- Aplicaciones científicas:
  - Para estimar y medir la biomasa en regiones acuáticas y el uso de etiquetas acuáticas para estudiar el ecosistema marino.
  - Medida de características del agua, olas y estado del mar.
  - Estudios topográficos del fondo y subsuelo marino. Permite también realizar arqueología subacuática.

CAPÍTULO 2. BASE TEÓRICA

2.1. Determinación de las coordenadas del blanco

En este apartado explicamos la base matemática necesaria para seguir el trabajo y los algoritmos implementados en el sistema.

2.1.1. Principios básicos de posicionamiento

2.1.1.1. Posicionado en dos y tres dimensiones

Para el posicionado en dos dimensiones, requerimos definir dos variables para definir un plano de trabajo y un origen de coordenadas. En este caso, empezaremos trabajando con X e Y, en el plano horizontal.

El posicionado en este plano puede realizarse en cartesianas, refiriéndonos al valor de X e Y en sus respectivos ejes respecto al origen de coordenadas; o en polares, conociendo una distancia (r) respecto al origen de coordenadas y un ángulo respecto a alguno de los dos ejes (α), en particular medido respecto al eje X.
En el siguiente esquema podemos apreciar en verde los datos referentes a las coordenadas cartesianas (X, Y) y en rojo las coordenadas polares (r, α).

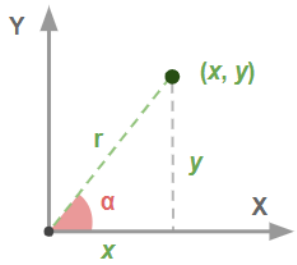

Fig. 2.1. Esquema posición bidimensional

En el caso de trabajar en tres dimensiones, necesitaríamos declarar una tercera variable para las coordenadas cartesianas (X, Y, Z) y un segundo ángulo para las coordenadas polares (r, α, β). En este caso, α es el ángulo de azimut definido respecto al eje X, y β el ángulo de azimut respecto al eje Z.

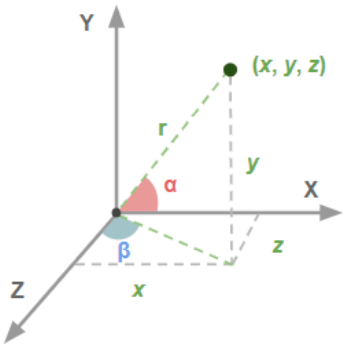

Fig. 2.2. Esquema posición tridimensional

2.1.1.2. Posicionado y triangulación de objetos mediante los sensores

Tal y como hemos visto en el subapartado anterior, para posicionar un objeto en dos dimensiones debemos calcular sus coordenadas X e Y. En este proyecto, utilizaremos sensores de ultrasonidos para obtener la distancia a la que se encuentra el objeto.

De esta manera, para poder posicionar un objeto en dos dimensiones, deberíamos disponer de al menos dos sensores de ultrasonidos. Conociendo la posición de los dos sensores y las distancias de lectura (d_1, d_2), somos capaces de triangular su posición y obtener las coordenadas (x, y) del objetivo. En el siguiente esquema se muestra la situación:

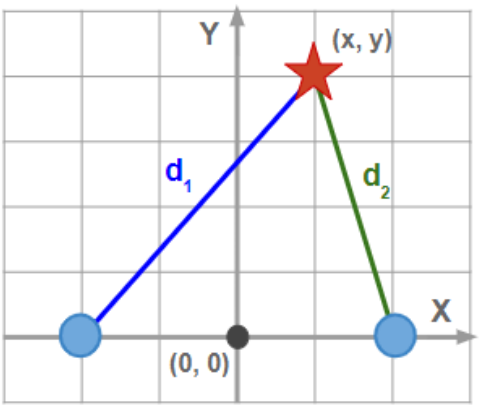

Fig. 2.3. Esquema de triangulación de posición

Debemos considerar que los sensores de ultrasonidos abarcan un haz de medida o zona de lectura limitada, así que para detectar y posicionar el objeto correctamente hemos de asegurar estar dentro del haz de medida de cada uno de los dos sensores. De lo contrario, no obtendríamos la lectura de una de las distancias y la triangulación del objeto no sería posible.

Como veremos más adelante, el sensor usado HC-SR04 tiene un haz angular de cobertura limitado. Si alineamos los sensores, el área superpuesta de la cobertura de ambos sensores será el área útil de trabajo posible para posicionar un objeto. Por este motivo, debemos tener en cuenta la distancia de separación entre sensores y calcular el área útil de trabajo teórico.

Por ejemplo, si disponemos los sensores en línea con el eje X y separados una distancia d la situación sería la siguiente:

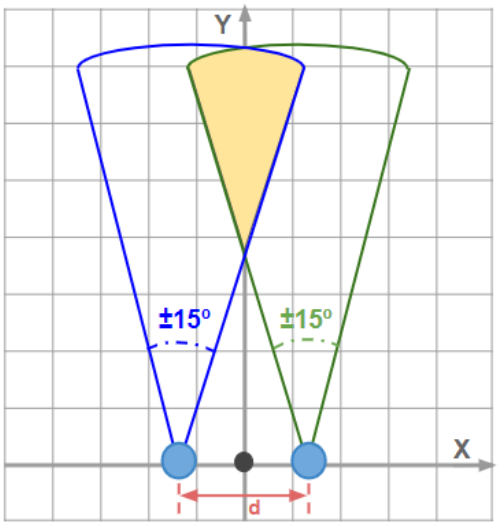

Fig. 2.4. Área útil de trabajo

En este caso, el área útil de trabajo resultante de la superposición de la cobertura de ambos sensores puede apreciarse en color amarillo.

Finalmente, con las distancias obtenidas por los sensores de ultrasonidos (d_1, d_2) y sabiendo la configuración de los mismos, posición y distancia entre ambos, es posible calcular la posición del objetivo en (x, y) siguiendo diferentes métodos y algoritmos matemáticos, descritos en el siguiente apartado.

2.1.2. Métodos y algoritmos de posicionamiento

2.1.2.1. Método de posicionamiento básico por triangulación

Este primer método se basa en el uso de trigonometría básica para calcular la posición del objetivo. La situación se resume en la siguiente figura:

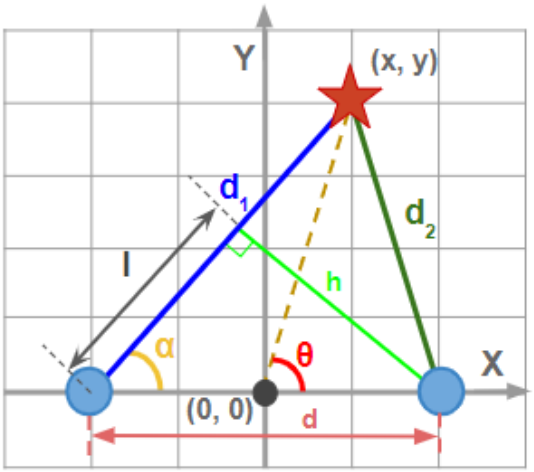

Fig. 2.5. Esquemático de triangulación básica

Empezamos analizando el triángulo inferior formado por $\boldsymbol{d}$, $\boldsymbol{l}$ y $\boldsymbol{h}$:

$$l^2 = d^2 - h^2 \quad \textbf{(2.1)}$$

Seguidamente analizamos el triángulo superior:

$$d_2{}^2 = (d_1 - l)^2 + h^2 = d_1{}^2 - 2ld_1 + l^2 + h^2 \quad \textbf{(2.2)}$$

$$d_2{}^2 = d_1{}^2 - 2d_1\sqrt{d^2 - h^2} + d^2 \quad \textbf{(2.3)}$$

Y aislamos el parámetro h:

$$\sqrt{d^2 - \left(\frac{d^2 + d_1{}^2 - d_2{}^2}{2d_1}\right)^2} = h \quad \textbf{(2.4)}$$

De esta manera, podemos calcular el ángulo α y la posición del objeto:

$$\alpha = \operatorname{asin}\left(\frac{h}{d}\right) \quad \textbf{(2.5)}$$

$$x = d_1 \cdot \cos(\alpha) - \frac{d}{2} \quad \textbf{(2.6)}$$

$$y = d_1 \cdot \sin(\alpha) \quad \textbf{(2.7)}$$

Y finalmente, podemos calcular el ángulo θ:

$$\tan(\theta) = \frac{y}{x} = = \frac{d_2 \cdot \sin(\alpha)}{d_1 \cdot \cos(\alpha) - d/2} \quad \textbf{(2.8)}$$

2.1.2.2. Método de posicionamiento por circunferencias

El siguiente método consiste en considerar que cada uno de los sensores crea una circunferencia centrada en sí mismo y con radio la distancia relativa al objetivo. De esta manera, para localizar el objetivo debemos resolver las ecuaciones y encontrar el punto de cruce entre ambas circunferencias.

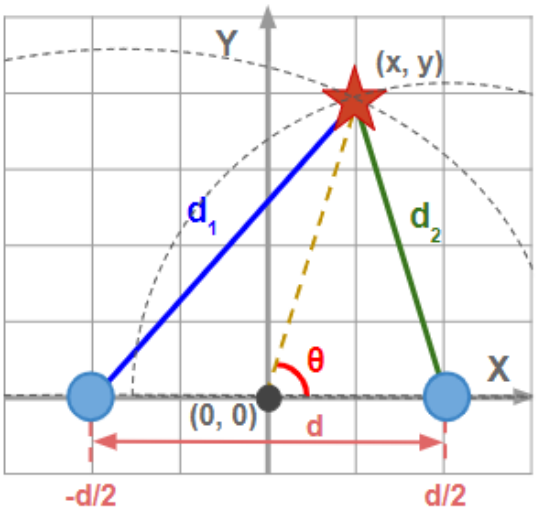

Fig. 2.6. Planteamiento método por circunferencias

Empezamos con las ecuaciones de las circunferencias que describen los sensores sabiendo que d_1 y d_2 son las distancias de lectura de cada sensor:

$$d_1{}^2 = y^2 + \left(x + \frac{d}{2}\right)^2 \quad \textbf{(2.9)}$$

$$d_2{}^2 = y^2 + \left(x - \frac{d}{2}\right)^2 \quad \textbf{(2.10)}$$

Sabiendo que la coordenada y es la misma en ambas ecuaciones, igualamos:

$$d_1{}^2 - \left(x + \frac{d}{2}\right)^2 = d_2{}^2 - \left(x - \frac{d}{2}\right)^2 \quad \textbf{(2.11)}$$

$$d_1{}^2 - d_2{}^2 = x^2 + xd + \frac{d^2}{4} - x^2 + xd - \frac{d^2}{4} \quad \textbf{(2.12)}$$

Obtenemos:

$$d_1{}^2 - d_2{}^2 = 2xd \quad \textbf{(2.13)}$$

Y aislamos la coordenada x del objetivo:

$$x = \frac{{d_1}^2 - {d_2}^2}{2d} \tag{2.14}$$

Y la coordenada y del mismo:

$$y = \sqrt{{d_1}^2 - \left(x + \frac{d}{2}\right)^2} \tag{2.15}$$

Finalmente podemos calcular la posición polar del objetivo, calculando el ángulo θ y la distancia r respecto al origen de coordenadas:

$$\theta = \text{arctg}\left(\frac{y}{x}\right) \tag{2.16}$$

$$r = \sqrt{x^2 + y^2} \tag{2.17}$$

2.1.2.3. Método de posicionamiento por elipses

En este caso, en lugar de considerar circunferencias, realizamos los cálculos teniendo en cuenta que el sensor entrega como observable el tiempo de vuelo del pulso (ida y vuelta) y el lugar geométrico, que representa las potenciales posiciones del blanco, es una elipse.

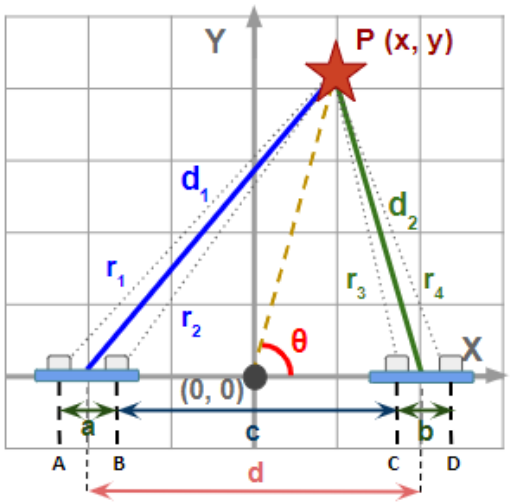

Fig. 2.7. Planteamiento método por circunferencias

Donde la medida de la pareja AB es $2d_1 = r_1 + r_2$ y la pareja CD es $2d_2 = r_3 + r_4$.

Empezamos escribiendo las ecuaciones de las elipses para cada uno de los sensores de ultrasonidos. Por un lado, tenemos la medida 1: $2d_1 = |AP| + |BP|$

$$\sqrt{\left(x + \frac{d}{2} + \frac{a}{2}\right)^2 + y^2} + \sqrt{\left(x + \frac{d}{2} - \frac{a}{2}\right)^2 + y^2} = 2d_1 \quad \textbf{(2.18)}$$

Y por el otro, tenemos la medida 2: $2d_2 = |CP| + |DP|$

$$\sqrt{\left(x - \frac{d}{2} + \frac{b}{2}\right)^2 + y^2} + \sqrt{\left(x - \frac{d}{2} - \frac{b}{2}\right)^2 + y^2} = 2d_2 \quad \textbf{(2.19)}$$

La posición del blanco P será la intersección de las dos elipses y asumiremos que el resultado es válido para toda y>0.

$$f_1(x,y) = \sqrt{\left(x + \frac{d}{2} + \frac{a}{2}\right)^2 + y^2} + \sqrt{\left(x + \frac{d}{2} - \frac{a}{2}\right)^2 + y^2} - 2d_1 \quad \textbf{(2.20)}$$

$$f_2(x,y) = \sqrt{\left(x - \frac{d}{2} + \frac{b}{2}\right)^2 + y^2} + \sqrt{\left(x - \frac{d}{2} - \frac{b}{2}\right)^2 + y^2} - 2d_2 \quad \textbf{(2.21)}$$

Linealizamos las ecuaciones **(2.20)** y **(2.21)** para un punto alrededor de P(x, y):

$$f_1(x,y) \approx f_1(x_0,y_0) + \frac{\partial f_1}{\partial x}(x - x_0) + \frac{\partial f_1}{\partial y}(y - y_0) \quad \textbf{(2.22)}$$

$$f_2(x,y) \approx f_2(x_0,y_0) + \frac{\partial f_2}{\partial x}(x - x_0) + \frac{\partial f_2}{\partial y}(y - y_0) \quad \textbf{(2.23)}$$

Podemos escribir el sistema de ecuaciones de manera simplificada:

$$A_1 = B_1\Delta x + C_1\Delta y \quad \textbf{(2.24)}$$

$$A_2 = B_2\Delta x + C_2\Delta y \quad \textbf{(2.25)}$$

$$A_1 = f_1(x,y); \quad B_1 = \frac{\partial f_1}{\partial x}; \quad C_1 = \frac{\partial f_1}{\partial y} \quad \textbf{(2.26)}$$

$$A_2 = f_2(x,y); \quad B_2 = \frac{\partial f_2}{\partial x}; \quad C_2 = \frac{\partial f_2}{\partial y} \tag{2.27}$$

Seguidamente calculamos las derivadas parciales

$$\frac{\partial f_1}{\partial x} = \frac{x + \frac{d}{2} + \frac{a}{2}}{\sqrt{\left(x + \frac{d}{2} + \frac{a}{2}\right)^2 + y^2}} + \frac{x + \frac{d}{2} - \frac{a}{2}}{\sqrt{\left(x + \frac{d}{2} - \frac{a}{2}\right)^2 + y^2}} \tag{2.28}$$

$$\frac{\partial f_1}{\partial y} = \frac{y}{\sqrt{\left(x + \frac{d}{2} + \frac{a}{2}\right)^2 + y^2}} + \frac{y}{\sqrt{\left(x + \frac{d}{2} - \frac{a}{2}\right)^2 + y^2}} \tag{2.29}$$

$$\frac{\partial f_2}{\partial x} = \frac{x - \frac{d}{2} + \frac{b}{2}}{\sqrt{\left(x - \frac{d}{2} + \frac{b}{2}\right)^2 + y^2}} + \frac{x - \frac{d}{2} - \frac{b}{2}}{\sqrt{\left(x - \frac{d}{2} - \frac{a}{2}\right)^2 + y^2}} \tag{2.30}$$

$$\frac{\partial f_2}{\partial y} = \frac{y}{\sqrt{\left(x - \frac{d}{2} + \frac{b}{2}\right)^2 + y^2}} + \frac{y}{\sqrt{\left(x - \frac{d}{2} - \frac{b}{2}\right)^2 + y^2}} \tag{2.31}$$

El problema puede ser resuelto matricialmente

$$\underline{A} = \underline{\underline{M}} \cdot \underline{\Delta \overline{x}} \tag{2.32}$$

$$\underline{A} = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}; \quad \underline{\underline{M}} = \begin{pmatrix} B_1 & C_1 \\ B_2 & C_2 \end{pmatrix}; \quad \underline{\Delta \overline{x}} = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{2.33}$$

$$\underline{\Delta \overline{x}} = \underline{\underline{M}}^{-1} \cdot \underline{A} \tag{2.34}$$

Donde:

$$\underline{\underline{M}}^{-1} = \begin{pmatrix} C_2 & -C_1 \\ -B_2 & B_1 \end{pmatrix} \cdot \frac{1}{B_1 C_2 - C_1 B_2} \tag{2.35}$$

Y finalmente obtenemos la posición (x, y) del blanco:

$$x = x_0 + \frac{C_2 A_1 - C_1 A_2}{B_1 C_2 - C_1 B_2} \quad \textbf{(2.36)}$$

$$y = y_0 + \frac{-B_2 A_1 + B_1 A_2}{B_1 C_2 - C_1 B_2} \quad \textbf{(2.37)}$$

2.2. Electrónica: sensores y motores

En este proyecto utilizaremos una placa Arduino como base de trabajo a la que conectaremos los sensores y los motores para obtener las lecturas de distancia y para realizar los movimientos del sistema de seguimiento oportunos.

2.2.1. Arduino

2.2.1.1. Fundamentos

Arduino es conocido por desarrollar placas electrónicas y gran cantidad de componentes electrónicos. Las placas electrónicas son tarjetas que integran un microcontrolador y un conjunto de pines de entrada y salida para conectar todo tipo de dispositivos. Usando una aplicación y un ordenador, podemos crear programas de control para interaccionar con el medio físico mediante sensores y actuadores.

El microcontrolador es un circuito integrado programable capaz de ejecutar las ordenes establecidas y almacenadas en su memoria. Los pines de Arduino puede ser entradas analógicas o digitales, o bien salidas digitales. Las entradas analógicas permiten leer valores de tensión de 0 a 5 V con una resolución característica del microcontrolador, por ejemplo, si dispusiéramos de 10 bits (1024) en el convertidor analógico/digital, las entradas analógicas tendrían una resolución aproximada de unos 5 mV, 5/1024.

Por otro lado, las entradas digitales se caracterizan por dos niveles de señal, el alto (HIGH – 5 V) y el bajo (LOW – 0 V), para medición de señales digitales. Hay salidas digitales que disponen de PWM (Pulse Width Modulation) o modulación por ancho de pulso, de esta manera pueden simular una salida analógica variando el ancho de los pulsos, modificando la frecuencia y el ciclo de trabajo de la señal.

2.2.1.2. Requisitos y componentes básicos

Para empezar con Arduino, necesitaremos obtener una placa Arduino, ciertos componentes electrónicos para montar nuestro sistema y un ordenador para crear nuestros programas de control.

Para decidir la placa Arduino a usar, deberemos considerar el número de pines de entrada/salida mínimos para conectar los elementos requeridos y la capacidad del microcontrolador a utilizar.

Entre las placas más conocidas encontramos [3]:

- Arduino UNO:
  Es una de las placas más básicas del mercado, basada en el microcontrolador ATmega328. Cuenta con 14 entradas/salidas digitales (6 de las cuales pueden ser PWM) y 6 entradas analógicas. Dispone de conector USB para el ordenador, conector de alimentación, una cabecera para conector serie y un oscilador cerámico de 16 MHz.

- Arduino LEONARDO:
  Es una placa prácticamente idéntica a Arduino UNO, pero basada en el microcontrolador ATmega32u4 y con comunicación USB integrada en lugar de comunicación serial, lo que permitiría usar funciones de ratón o teclado directamente.

  Algo que puede resultar destacable, es que tanto en la Arduino UNO como la LEONARDO, es posible configurar las entradas analógicas como entradas/salidas digitales usando la función `pinMode()` de Arduino.

- Arduino MEGA:
  Es una placa usada para proyectos complejos. A diferencia de la anterior, cuenta con 54 entradas/salidas digitales (15 de las cuales pueden ser PWM), 16 entradas analógicas y 4 conectores para puerto serial.

Los precios oficiales de las placas rondan los 20-30€, aunque es posible encontrarlas online a un precio inferior si renunciamos a la última versión lanzada. Por el número de pines necesarios en este proyecto, para el sistema bidimensional usamos la UNO y para el tridimensional la MEGA.

2.2.2. Sensores de ultrasonidos

2.2.2.1. Principios básicos

Los sensores de ultrasonidos son dispositivos detectores de proximidad que detectan objetos situados en un rango de pocos centímetros a varios metros.

Estos sensores trabajan usando el aire o el agua como medio y se basan en un emisor que emite una señal sonora, y un receptor, que mide el tiempo que tarda la señal en regresar. De esta manera, sabiendo el tiempo que la señal ha necesitado para viajar en ida y vuelta, y la velocidad de propagación de la misma en el medio característico, es posible calcular la distancia al objeto que provoca la reflexión.

Estos sensores no requieren de contacto físico directo con el objeto a detectar (a diferencia de sensores mecánicos o pulsadores), la única condición necesaria es que el objeto sea de un material reflector de las ondas sonoras.

2.2.2.2. Estudio de mercado

El mercado de sensores de ultrasonidos es muy amplio, desde los más sencillos y baratos a los más complejos y caros. Por un lado, tenemos los sensores de ultrasonidos de alcance ajustable (utilizados como interruptores NPN o PNP), no interesantes en este proyecto; y los sensores con salida digital o analógica usados para mediciones de distancia. Estos son algunos ejemplos de sensores por ultrasonidos de distancia:

Tabla 2.1. Gama UMXX de Sick AG [4]

Modelo	UM12	UM18	UM30
Alcance máximo [mm]	350	1300	8000
Resolución [mm]	0,069	0,069	0,18
Precisión de repetición [%]	±0,15	±0,15	±0,15
Voltaje de trabajo [V]	0-10	0-10	0-10
Precio orientativo [€]	180	195	285

- Sensor HC-SR04 de Arduino: sensor de ultrasonidos que opera en rangos de 3 cm a 400 cm con una precisión de 3 mm y un haz de medida de ±15°. El precio online está alrededor de 2 €. Debido a las especificaciones y precio, es el utilizado en este proyecto; en el siguiente subapartado se proporcionan más detalles de sus características.

Por otro lado, también sería interesante comparar con sensores láser:

Tabla 2.2. Gama DXXX de Sick AG [4]

Modelo	DX35	DX50	DX100
Alcance máximo [m]	35	20	300
Resolución [mm]	10	3	5
Precisión de repetición [mm]	0,5 - 5	0,25 - 5	0,5 - 5
Voltaje de trabajo [V]	0-10	0-10	0-10
Precio orientativo [€]	270	495	615

- Sensor VL53L0X de Arduino: sensor infrarrojo láser capaz de detectar objetos en un rango entre 5 cm y 200 cm con una precisión de 0,3 mm. La precisión es superior a los sensores de ultrasonidos y no depende de las condiciones ambientales ni posibles ecos. El inconveniente principal es que el haz de medida es muy estrecho, de manera que solo detectaría objetos frente del sensor, y no sería posible detectar obstáculos. El precio orientativo ronda los 5 €.

A la hora de elegir un sensor, debemos basarnos en el rango de operación y precisión de medida requerida, en el voltaje de trabajo y en el precio. En el mundo industrial y profesional se utilizan sensores ultrasónicos y de infrarrojos o láser considerablemente sofisticados y caros.

Según los requisitos de este proyecto, nos interesaría reducir los costes lo máximo posible, así que los dos únicos modelos restantes serían el sensor HC-SR04 y el VL53L0X.

Aunque la precisión del sensor infrarrojo VL53L0X es notoriamente superior que la del sensor ultrasónico HC-SR04, el sensor láser tiene un haz de trabajo muy reducido (prácticamente reducido a un punto), lo que no permite detectar obstáculos, por lo tanto, queda descartado para la detección y seguimiento de objetos.

2.2.2.3. Sensor de ultrasonidos HC-SR04

El sensor HC-SR04 permite medir distancias mediante ultrasonidos. El rango de medida teórico del sensor HC-SR04 es de 3 cm a 400 cm, con una resolución aproximada de 3 mm, dentro de un haz angular máximo de 30º (±15º).

El conexionado del sensor está compuesto por cuatro pines: Vcc (voltaje de alimentación a 5 V), Trig (trigger o input del sensor), Echo (output del sensor) y GND (tierra o punto de referencia a 0 V). Los pines Trig y Echo deben ir conectados a salidas/entradas digitales respectivamente.

Fig. 2.8. Sensor de ultrasonidos HC-SR04 [5]

El funcionamiento del sensor es el siguiente: si aplicamos un pulso de 5 V durante un mínimo de 10 us por el puerto Trigger, el sensor emite 8 pulsos ultrasónicos a una frecuencia de 40 kHz [6]. De esta manera, el pulso reflejado puede ser detectado, generando una señal en alto (de 5V). Esta señal será un pulso rectangular de ancho equivalente al retardo entre el pulso emitido y el recibido por reflexión.

Finalmente, con el tiempo de retardo podemos obtener la distancia aplicando la siguiente ecuación:

$$d[cm] = \frac{v_{prop}\left[\frac{cm}{s}\right] \cdot t_{retardo}[s]}{2} \qquad \textbf{(2.38)}$$

Donde v_{prop} es la velocidad de propagación del sonido en el aire, la cual puede determinarse conociendo el coeficiente adiabático γ y la masa molecular M del

aire, la constante del gas R y la temperatura del medio T, con la siguiente expresión:

$$v_{prop}[cm/s] = \sqrt{\frac{\gamma RT}{M}} \quad \textbf{(2.39)}$$

Siendo $\gamma = 1{,}4;\ R = 8{,}314\ ;\ \ T = 293{,}15\ K\ (20℃)\ \ y\ \ M = 0{,}029\ \frac{kg}{mol}\frac{J}{K \cdot mol}$

2.2.3. Motores de corriente continua

2.2.3.1. Motores de continua con codificador

Es posible encontrar motores de continua muy sofisticados que incluyen de manera interna o externa un codificador capaz de contar un gran número de eventos por revolución.

La característica de estos motores es una relación de transmisión muy alta entre el codificador y el eje de giro del motor que permite controlar y realizar pasos con una precisión muy alta. Además, el par motor y la velocidad de giro de estos motores es muy competitivo.

Sin duda, son una gran opción para proyectos complejos y exigentes debido a sus características, pero con un gran inconveniente: el precio elevado. Los motores de este tipo más destacados son de Actobotics [7], y los precios parten de 50€ sin considerar el controlador, que sumaría unos 10€ más al coste. Podemos encontrar algunos ejemplos en el apartado 2.1.3.4.

2.2.3.2. Servomotores

Un servomotor (servo) es un tipo de motor de continua con capacidad de control de posición. Un servo está compuesto por un motor de continua común más una combinación de engranajes y un potenciómetro con tarjeta de control.

Por un lado, los engranajes permiten variar la relación de giro aumentando la precisión y el par motor; y el potenciómetro permite localizar la posición del eje de giro.

Los servomotores pueden ser de rango limitado, por ejemplo, a 180 grados o de rotación continua, pudiendo superar los 360 grados. Los servos de rango limitado permiten variar el ángulo de rotación y un servo de rotación continua, la velocidad de rotación.

Además, los servomotores de rango limitado se controlan mediante una señal de control modulada, como puede ser la PWM para modificar el ángulo según un cierto valor de voltaje. El potenciómetro interno del servomotor forma un divisor de tensión que permite ajustar y conocer el ángulo del eje del motor según el voltaje resultante.

2.2.3.3. Motores paso a paso

Un motor paso a paso (*stepper*) también es un motor de continua pero que realiza pasos incrementales angulares al aplicarle un pulso o señal eléctrica, éstos son controlados mediante la excitación de las bobinas de manera secuencial. Los motores paso a paso se caracterizan por dos parámetros principales: la resolución de paso [pasos/rev] y la velocidad de giro [rev/min].

En este caso, nos centraremos en los motores paso a paso unipolares que constan de dos bobinas con un punto medio común. De esta manera, del *stepper* salen un total de 5 cables, tenemos 2 cables por bobina más el cable común, como puede apreciarse en la siguiente imagen:

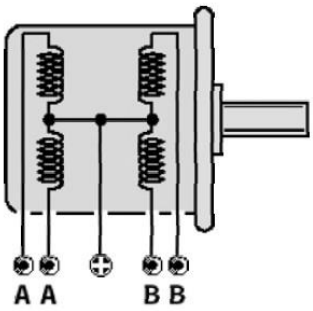

Fig. 2.9. Esquemático bobinado motor [8]

Al tener el punto medio común, podría considerarse que realmente hay 4 bobinas diferentes, por lo tanto 4 fases de excitación posibles.

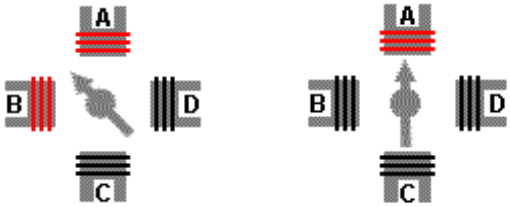

Fig. 2.10. Secuencia de excitación (en rojo) normal y de paso completo [9]

La secuencia de excitación de las bobinas permite realizar el paso a paso angular del motor y existen 3 maneras posibles de efectuar la secuencia:

- Secuencia normal: el motor realiza un paso por excitación ya que las bobinas se activan de dos en dos.
- Secuencia de paso completo: únicamente hay una bobina activa en el mismo instante y el eje gira hacia la bobina activa. El consumo y el par motor sería inferior en esta configuración.
- Secuencia de paso medio: es una combinación de los casos anteriores, consiguiendo realizar medio paso por cada excitación. Hay instantes con dos bobinas activas e instantes con solo una.

La secuencia de excitación escogida en este proyecto podría ser, por ejemplo, la opción de paso medio, para así conseguir doblar la precisión de paso del

motor. En el apartado 2.3.1.3 se determina cómo realizar y declarar la secuencia de excitación para mover el motor con el programa de control.

2.2.3.4. Estudio de mercado

A continuación, podemos empezar analizando los servomotores. Los servomotores suelen clasificarse en nano, *micro*, *mini* o *large* según su tamaño y par motor. Podemos apreciar una tabla comparativa de algunos modelos de Hitec:

Tabla 2.3. Gama de servomotores Hitec [10]

Modelo	HS-40	HS-81	HS-225BB	HS-755HB
Tipo	Nano	Micro	Mini	Large
Par máximo [kg-cm]	0,75	2,95	4,85	13,2
Velocidad máxima [s/60º]	0,10	0,10	0,12	0,25
Precio orientativo [€]	8	13	15	34

También sería posible usar motores de continua con codificador integrado por ejemplo:

Tabla 2.4. Motores de continua con codificador [7]

Modelo	638288	638290	638296	638298
Resolución codificador	5462	4435	2774	1245
Par máximo [kg-cm]	42	34	21	11.5
Velocidad máxima [rpm]	26	32	52	116
Precio orientativo [€]	50	50	50	50

Este tipo de motores son los más competitivos y con mejores características del mercado para realizar proyectos complejos, siempre y cuando el precio no sea un inconveniente. Por ejemplo, tener una resolución de codificador de 3600 pasos significa, que podríamos obtener una precisión angular de $\frac{360°}{3600} = 0{,}1°$; es decir, una precisión de paso realmente buena. Y finalmente, algunos de los motores paso a paso existentes en el mercado:

Tabla 2.5. Motores paso a paso [11]

Modelo	HT08	HT11	HT17	HT34
Resolución de paso [º]	1	1.5	1.8	1.8
Par máximo [kg-cm]	0,85	1,1	4,80	13,2
Velocidad máxima [s/60º]	0,10	0,10	0,12	0,25
Precio orientativo [€]	7	12	15	27

Por último, el motor 28BYJ-48: es un motor paso a paso con una precisión de paso de 5,625 grados, pero con una reductora 1/64 que permite mejorar el paso a 0,088 º, alcanzando un total de 4096 pasos por vuelta. La velocidad de giro máxima es de 8 revoluciones por minuto (rpm) y el par motor de 0,35 kg/cm. Pero

sin duda, el gran aliado es el precio, ya que podemos comprarlo por menos de 5€ con el controlador incluido.

Comparando los servomotores y los motores paso a paso, cabe resaltar que si lo que nos interesa es precisión angular de giro, sin necesidad de un par y velocidad de giro relativamente altas e intentando reducir costes, los motores paso a paso serían la opción a elegir.

2.2.3.5. Motor paso a paso 28BYJ-48

Finalmente, nos decidimos por usar el motor paso a paso 28BYJ-48. El motor 28BYJ-48 alcanza un paso incremental de 0,088º (4096 pasos por vuelta).

Después de realizar pruebas con el motor paso a paso, podemos observar una resolución de 2048 pasos por vuelta (resolución gradual de 0,176º) para alcanzar velocidad de giro máxima de 8 revoluciones por minuto (8 rpm). Si incrementamos la resolución de giro, la velocidad de rotación disminuye bruscamente y el movimiento no resulta aparentemente fino (aumenta la vibración del mismo).

Por otro lado, no interesa perder resolución de giro más de lo necesario y así garantizar exactitud y precisión angular de apuntamiento. De esta manera, encontramos la resolución de 4096 pasos por vuelta que otorga la máxima resolución (0,088º), aunque no asegure la velocidad máxima.

Como se ha explicado en el apartado anterior, el motor 28BYJ-48 consta de 5 cables y requiere el uso del controlador ULN2003 para conectarlo a la placa Arduino y realizar el control y excitación de las bobinas.

Fig. 2.11. Controlador ULN2003 [9]

El controlador ULN2003 consta de un conector blanco con 5 pines para el motor, 4 pines para excitar las bobinas y dos pines de alimentación. También es posible alimentarlo a 12 V si fuera necesario. Cabe destacar que el pin de alimentación no utilizado debe estar puenteado.

Cada uno de los 4 pines (IN1, IN2, IN3, IN4) irán conectados a una salida digital de la placa Arduino, y los pines de alimentación a +5 V y a GND respectivamente. La secuencia de excitación de estos pines se describe en el apartado 2.3.1.3.

2.3. Herramientas de programación

En este apartado encontraremos la información necesaria para la programación y el desarrollo del sistema de control del sonar.

2.3.1. Arduino

2.3.1.1. Principios básicos de programación Arduino

El código Arduino facilita la programación de un microcontrolador en un entorno sencillo y con gran potencial. Arduino IDE (© Arduino, 2019) es una aplicación que permite escribir, ordenar y cargar el código directamente en el microcontrolador de la placa Arduino en un lenguaje de programación basado en C++.

Un programa Arduino se escribe en un fichero (o *sketch*) con extensión *.ino* que debe incluir al menos la función `setup`(), que se ejecuta cuando se inicia el programa, y `loop`(), que se ejecuta repetitivamente. De esta manera, todo aquello que consideremos inicialización de código debe estar incluido en `setup`() y las líneas que han de ejecutarse de manera iterada debe incluirse en `loop`().

En `setup`() es necesario inicializar los pines para declararlos entrada o salida digital usando la función `pinMode`(pin, modo) con el número de pin a configurar y el modo si es entrada (INPUT) y si es salida (OUTPUT). En el caso de los pines digitales será posible ponerlos en modo alto (+5 V) o en bajo (+0 V), para ello disponemos de la función `digitalWrite`(pin, modo) donde el modo sería alto (HIGH) o bajo (LOW).

El fichero principal debe estar siempre contenido en una carpeta con el mismo nombre que el fichero y puede contener o llamar otros archivos usando la directiva `#include` con el nombre o dirección del archivo.

2.3.1.2. Control del sensor de ultrasonidos HC-SR04

Conociendo los principios de funcionamiento del sensor HC-SR04 mencionados en la sección 2.2.2.3 y recordando la configuración de los pines del sensor:

- Vcc: +5 V
- GND: referencia (+0 V)
- Trig: salida digital
- Echo: entrada digital

Una vez realizado el conexionado del sensor a la placa Arduino, por ejemplo, conectando el *Trigger* al pin 8 y el pin *Echo* al pin 9, el programa básico de control para obtener el retardo de llegada del eco sería el siguiente:

```
int TRIG = 8;
int ECHO = 9;
```

```
  setup(){

pinMode(TRIG, OUTPUT); //Salida digital
pinMode(ECHO, INPUT);  //Entrada digital

  loop(){

  //Desactivamos TRIGGER
  digitalWrite(TRIG, LOW);
  delayMicroseconds(10);

  //Activamos TRIGGER -> EMITIMOS PULSO
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  //Tiempo que ECHO está en estado HIGH
  float tiempo = pulseIn(ECHO, HIGH);

  const float vprop = sqrt(1.4*8.314*293.15/0.029)*100; //cm/s

  float distancia = tiempo* vprop / 1000000 / 2;

  }
```

Una vez hemos obtenido el tiempo, aplicamos la fórmula **(2.38)** con la velocidad de propagación del medio para calcular la distancia.

Una alternativa a calcular la velocidad de propagación sería obtener el tiempo de retardo conociendo la posición exacta del objetivo y extraer la relación entre ambas para calibrar el sistema.

2.3.1.3. Control del motor paso a paso 28BYJ-48

Ahora sabemos que para el control de un motor paso a paso se requiere definir una secuencia de excitación de las bobinas para realizar un incremento angular.

Una opción de definir estas secuencias sería usando una matriz, donde cada columna sea el estado de la bobina (A, B, C y D) y cada fila el instante o fase de la excitación (4 o 8 estados, este último en el caso de medios pasos).

En este caso, podríamos definir cada una de las secuencias comentadas en la sección 2.2.3.3:

- Secuencia normal:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

- Secuencia de paso completo:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Secuencia de medio paso:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

En las matrices anteriores, un 1 significa que la bobina debe estar en alto (HIGH) con voltaje aplicado de 5V y un 0 que la bobina debe estar en bajo (LOW) sin voltaje. En este caso, usaríamos la secuencia de pasos medios para que el movimiento resulte los más fino y preciso posible, la matriz en código la definiríamos tal que:

```
//Secuencia de excitación de las bobinas
int sec [8][4] =
{
  {1, 0, 0, 0},
  {1, 1, 0, 0},
  {0, 1, 0, 0},
  {0, 1, 1, 0},
  {0, 0, 1, 0},
  {0, 0, 1, 1},
  {0, 0, 0, 1},
  {1, 0, 0, 1}
};
```

Seguidamente, debemos calcular la ratio de pasos realizado por excitación. Como el motor tiene 4 bobinas, se necesitan 512 pasos para completar una vuelta con la reductora 1/64 y usando la secuencia de medio paso se requieren dos impulsos para realizar un paso; la ratio entre impulsos y grados sería:

$$\frac{4\ bobinas \cdot 512\ pasos \cdot 2\frac{impulsos}{pasos \cdot bobina}}{360°} = \frac{4096\ impulsos}{360°} \quad \textbf{(2.40)}$$

De esta manera, el número total de pasos o impulsos necesarios para girar un cierto ángulo θ sería:

$$Npasos = \frac{4096\ impulsos}{360°} \cdot \theta[°] \quad \textbf{(2.41)}$$

Con la secuencia definida y la ratio entre ángulo y pasos conocida, podríamos crear una función para mover el motor:

```
void moveMOTOR(double grados)
{
  float ratio = 4096/360;
  float tot_pasos = abs(grados)*ratio;
  int i = 0;

for (int pasos = 0; pasos < tot_pasos; pasos++)
    {
      if (grados>=0){
          digitalWrite(pinIN1, sec[i][0]);
          digitalWrite(pinIN2, sec[i][1]);
          digitalWrite(pinIN3, sec[i][2]);
          digitalWrite(pinIN4, sec[i][3]);
        }
        else{
          digitalWrite(pinIN1, sec[i][3]);
          digitalWrite(pinIN1, sec[i][2]);
          digitalWrite(pinIN1, sec[i][1]);
          digitalWrite(pinIN1, sec[i][0]);
        }

        i++;
        if (i==8)
        i = 0;

      delay(2); //Paso de excitación cada 2 ms
    }
}
```

La función `moveMotor`(double grados) mueve el motor los grados indicados en el parámetro entrada recorriendo la matriz fila por fila (excitando las bobinas conectadas en los pines `pinIN1`, `pinIN2`, `pinIN3` y `pinIN4`) hasta el alcanzar el número de pasos necesario. Así, por ejemplo, en la fila {1, 0, 0, 0} la bobina 1 estaría activa (+5 V) y las demás desactivadas (0 V).

2.3.2. Matlab

2.3.2.1. Introducción a Matlab

Matlab [13] es un entorno de trabajo pensado para el desarrollador de algoritmos que requieran cálculos numéricos y el análisis de los mismos en el campo de la ingeniería y de la ciencia. Matlab ofrece una gran variedad técnicas de manipulación y representación datos, implementación de funciones y algoritmos y creación de interfaces de usuario. Además de comunicación con otros programas y hardware. Basado en un lenguaje propio (lenguaje M) que permite realizar operaciones de vectores y matrices, gráficos, cálculos avanzados y programación orientada a objetos.

Matlab permite control y obtención de datos directa de placas de Arduino o lectura de datos desde puerto serie. Cualquiera de las opciones es posible para obtener los datos del microcontrolador y procesarlos.

2.3.2.2. Obtención y procesado de datos

En este proyecto utilizaremos Matlab para obtener todos los datos de interés de la placa Arduino, leyendo por el puerto serie (COM). De esta manera Matlab nos permitirá realizar todos los cálculos y gráficos necesarios para realizar el estudio completo de los sensores y del sistema de seguimiento. Las rutinas básicas utilizadas se encuentran el apéndice B.1.

2.3.3. Visual Studio Code

2.3.3.1. Introducción a Visual Studio Code

Visual Studio Code [14] es un editor de código fuente libre de Microsoft para desarrollar una gran variedad de proyectos de software. El uso de Visual Studio Code es recomendado también para proyectos Arduino ya que incluye una gran oferta de soporte para optimización y depuración de código (revisión y corrección de errores). Por otro lado, permite definir un entorno de trabajo que facilita la interacción entre varios ficheros y programas, trabajando en cualquier lenguaje de programación e incluso combinarlos.

En el caso de este proyecto, por semejanza a Arduino, trabajaremos en C++ orientado a objetos con clases. Una clase es una estructuración de código que permite agrupar variables, procedimientos y funciones, creando objetos con características e instancias propias.

Las clases deben definirse en una pareja de ficheros con extensión .h y .cpp. El fichero .h (cabecera) debe contener la estructura de la clase y las declaraciones de atributos, variables, procedimientos y funciones de la misma. El fichero .cpp (compilador) debe contener el desarrollo de todas las funciones declaradas en la clase.

Seguidamente vamos a mostrar un ejemplo definiendo una clase tipo ultrasonido. El fichero .h (Ultrasonido.h) debería seguir una estructura semejante:

```
#ifndef ULTRASONIDO_H
#define ULTRASONIDO_H

class Ultrasonido
{
 public: //ATRIBUTOS PÚBLICOS
    //PINES DEL SENSOR
    int GND = 2;
    int ECHO = 3;
    int TRIG = 4;
    int VCC = 5;

    //PROCEDIMIENTOS Y FUNCIONES
    double dameDISTANCIA();
    void ponPINES(int GND, int ECHO, int TRIG, int VCC);

};

#endif
```

En el fichero .h definimos los atributos de la clase ultrasonido, por ejemplo: los pines de conexionado, y los diferentes procedimientos y funciones, en este caso, para calcular la distancia leída por el sensor y modificar los pines.

A continuación, debemos crear las funciones declaradas en la clase ultrasonido, para ello usamos el fichero .cpp (Ultrasonido.cpp):

```
#include "Ultrasonido.h"

double Ultrasonido::dameDISTANCIA()
{

  double tiempo = getTIME(this->TRIG, this->ECHO); //tiempo de vuelo [µs]
  double  vprop = sqrt(1.4*8.314*293.15/0.029); //velocidad [m/s]

  return  tiempo*vprop/10000/2; //distancia [m]
}

void Ultrasonido::ponPINES(int gnd, int echo, int trig, int vcc)
{
   this->GND = gnd;
   this->ECHO = echo;
   this->TRIG = trig;
   this->VCC = vcc;
```

```
}
```

Como vemos, el fichero cpp debe incluir y llamar al fichero cabecera usando `#include`: “Ultrasonido.h”.

Una vez creada la clase Ultrasonido, podríamos crear objetos del tipo ultrasonido en otro fichero. Para poder usar la clase de ultrasonido en un programa Arduino (de extensión .ino) deberemos crear un archivo de configuración que permitirá enlazar el sketch principal con la clase y funciones creadas. Ejemplo (Ultrasonidos_conf.h):

```
#include "Ultrasonido.h"

Ultrasonido sensor1;

void iniciaSensor(int gnd, int echo, int trig, int vcc)
{
  sensor1.ponPINES(gnd, echo, trig, vcc);

  pinMode(this->TRIG, OUTPUT);
  pinMode(this->ECHO, INPUT);

  pinMode(this->VCC, OUTPUT);
  pinMode(this->GND, OUTPUT);

  digitalWrite(this->VCC, HIGH);
  digitalWrite(this->GND, LOW);
}

double dameDistancia()
{
   double dist1 = sensor1.dameDISTANCIA(3, 4);
}
```

En este último ejemplo, hemos creado un sensor de ultrasonidos con sus respectivos pines y calculamos la distancia a un objetivo. De esta manera, en un sketch de Arduino podríamos llamar a la función `iniciaSensor()` en el `setup()` para inicializar el sensor y a la función `dameDistancia()` en el `loop()` para obtener la distancia leída por el sensor.

2.3.3.2. Integración del sistema completo

El sistema completo del seguidor sonar (*tracker*) deberá organizarse en diferentes archivos para ordenar y optimizar el código, usando clases.

Por un lado, tendremos el fichero que ejecutará el microcontrolador de la placa de Arduino (fichero de extensión. ino) y por el otro lado, las clases que se consideren necesarias para el proyecto (ficheros .h y .cpp).

Archivo Arduino

- *sistemaCOMPLETO.ino*: programa principal que será ejecutado por el microcontrolador de Arduino. Ante la complejidad del código y la necesidad de declarar funcionalidades, se podría dividir en las siguientes partes:

    - Ultrasonido: Para usar un sensor de ultrasonidos.
    - Maths: Activa el módulo de cálculo y algoritmos.
    - Motor: Para activar un motor.

    - Tracker2D: Activa los sensores y el motor del seguidor sonar bidimensional (tracker2D).
    - Tracker3D: Activa los sensores y los motores del seguidor sonar tridimensional (tracker3D).
    - LED: Define y enciende un LED láser.

En la placa Arduino podemos conectar todos los sensores y motores necesarios y después decidir cual nos interesaría iniciar y utilizar, activando o desactivando cualquiera de las funcionalidades anteriores. En este caso, por ejemplo, el Tracker2D activaría dos sensores de ultrasonidos (declarados como sensor 1 y sensor 2) y un único motor. En cambio, el Tracker 3D emplearía cuatro sensores y dos motores.

Clase Tracker

Esta clase contiene la configuración de pines y funciones de control de los ultrasonidos y los motores que constituyen el tracker2D y el tracker3D. La constituyen los siguientes archivos:

- *Tracker.h*: Declaración de pines, variables y funciones del tracker.
- *Tracker.cpp*: Funciones del tracker

Clase Maths

Esta clase contiene todas las funciones y algoritmos matemáticos utilizados para calcular la posición del objetivo. La constituyen los siguientes archivos:

- *Maths.h*: Declaración de variables y funciones matemáticas.
- *Maths.cpp*: Funciones matemáticas.

Además, para enlazar las clases creadas con el archivo Arduino es aconsejable crear otros dos archivos, uno para instar el objeto tracker y el otro para crear el módulo maths.

- *Tracker_conf.h*: Fichero para llamar las funciones del tracker.
- *Maths_conf.h*: Fichero para llamar las funciones matemáticas.

Una vez definidos el seguidor sonar bidimensional y tridimensional en los siguientes capítulos, en el apéndice D.1. profundiza en el código del sistema de control.

CAPÍTULO 3. Estudio del sensor HC-SR04

3.1. Sistema de medida

En primer lugar, debemos estudiar el comportamiento y las limitaciones reales del sensor de ultrasonidos HC-SR04.

3.1.1. Primeras medidas

Para ello, realizamos un montaje que más adelante también servirá para probar el seguidor bidimensional, aunque de momento sólo utilizaremos un único sensor (el sensor derecho en el montaje mostrado en la figura 3.1.):

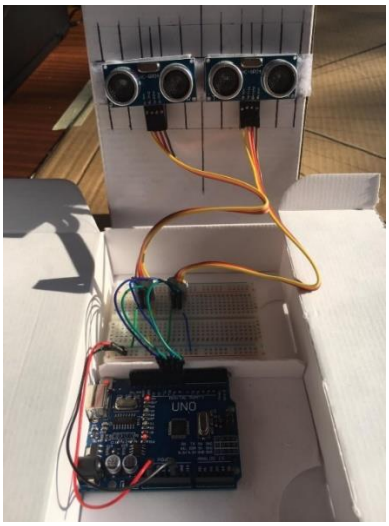

Fig. 3.1. Montaje preliminar para estudiar el sensor HC-SR04

Para realizar las medidas es aconsejable estar en un ambiente lo más controlado posible para evitar reflexiones (ecos) e interferencias externas. Además, para comprobar y verificar las medidas necesitaríamos una base graduada para conocer con exactitud la posición del objetivo a estudiar.

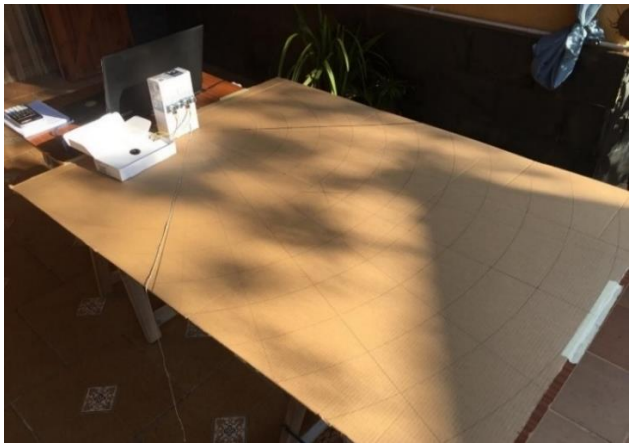

Fig. 3.2. Escenario de estudio con mesa graduada de medida

Realizamos el conexionado de un sensor de ultrasonidos a la placa de Arduino y usamos el programa de control del subapartado 2.3.1.2. Al programa deberíamos añadirle la siguiente línea en el `setup()`:

```
Serial.begin(9600);
```

E imprimir por el puerto serie el resultado obtenido para:

```
//OUTPUT del serial: distancia (radio)
  Serial.println(distancia, 2); //distancia con 2 decimales [cm]
```

Desde Matlab podremos leer el valor del puerto serie, guardarlo y procesarlo. En resumen, las medidas se han realizado siguiendo el mismo procedimiento:

- Obtención de los datos en Arduino e impresión por el puerto serie COM.
- Desde Matlab, lectura del puerto serie COM y procesado de los datos.
- En Matlab, obtenemos 10 lecturas de las distancias y calculamos el promedio y el error.
- El objetivo es una caja metálica de 9,5 x 9 cm.
- Para medidas < 50 cm, medimos con el lateral del objeto de 9,5 x 4,5 cm.

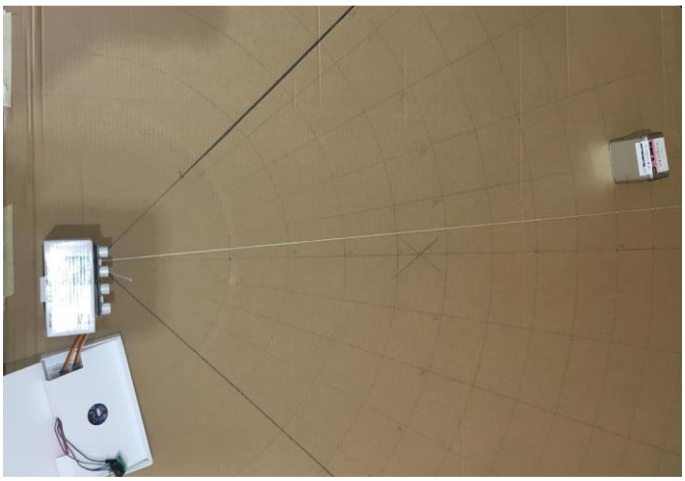

Fig. 3.3. Ejemplo de medida a 100 cm y -10º

Observando las primeras medidas, podemos comprobar que el sensor de ultrasonidos realiza medidas con un error inferior a 1,5 cm dentro de un haz angular aproximado de -5° a 5° (10°).

Fuera de ese rango angular, a distancias inferiores a 100 cm, el error aumenta considerablemente, hasta 2 cm de incertidumbre. En distancias de 70 cm a 120 cm el haz angular aumenta de -7,5° a 7,5° (15°). En el siguiente apartado se realiza un análisis más profundo de los resultados obtenidos tras las primeras medidas.

3.2. Caracterización del sensor

En el siguiente apartado analizamos las medidas obtenidas para caracterizar el sensor de ultrasonidos. En el apéndice B se encuentran las rutinas necesarias para la obtención y el procesado de datos, y podemos encontrar los resultados del estudio en el apéndice A.

3.2.1. Estudio de errores

Seguidamente, realizamos un estudio completo de las medidas calculando el error entre el valor obtenido y el valor real conocido de la posición del objetivo a partir de la distancia de la mesa graduada. Para ello calculamos el error relativo y absoluto de las medidas de 5 a 120 cm entre un ángulo de operación de -10º a 10º. Para facilitar la lectura y análisis de datos de la Tabla A.2 y de la Tabla A.3 del apéndice, se realizan los siguientes diagramas:

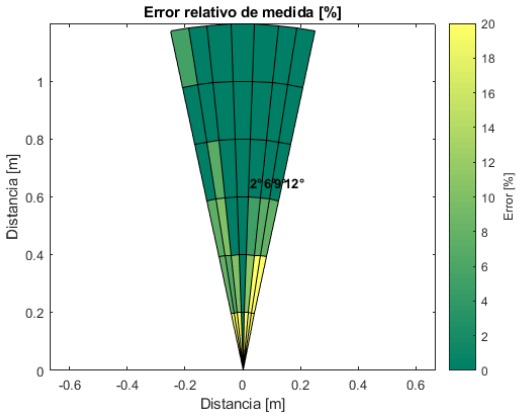

Fig. 3.4. Error relativo de medida

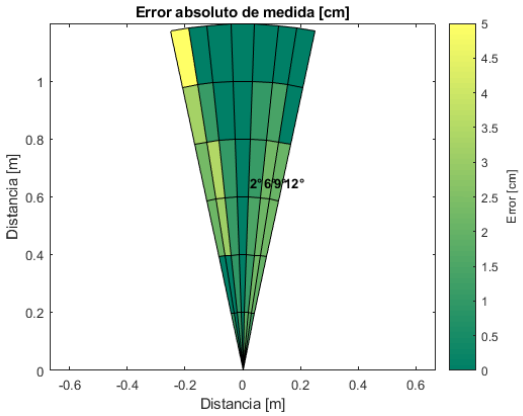

Fig. 3.5. Error absoluto de medida

En el caso del error relativo, las medidas superan un error del 15% solo para los casos de tener el objetivo a menos de 50 cm fuera de la perpendicular del sensor. En el caso del error absoluto, el peor de los casos sucede colocando el objetivo a -10º y provocando un error de medida de hasta 3 cm.

De esta manera, observando ambos diagramas, podemos concluir que el sensor obtiene buenos resultados siempre que el objetivo se localice a una distancia superior de 50 cm y no superemos los 10º de apertura angular. De esta manera, el haz operativo se reduce de los 30º teóricos a 20º reales.

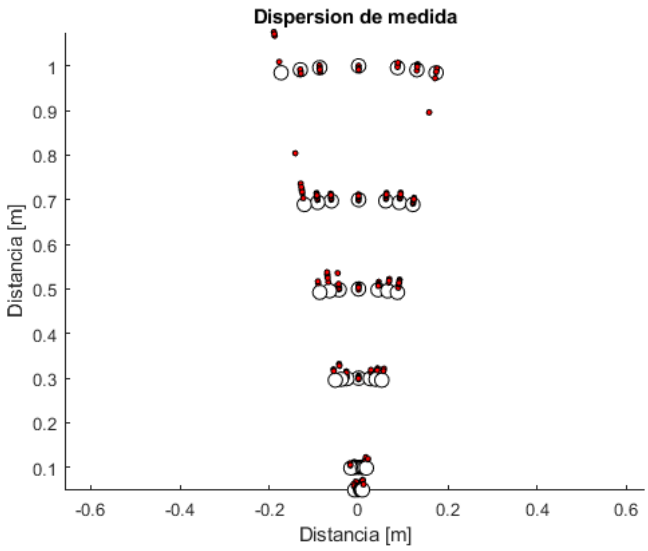

Fig. 3.6. Dispersión de medida del sensor

En la figura anterior, las medidas reales son los puntos rojos y todas aquellas que se encuentran fuera de los círculos negros superan 3 cm de error absoluto. En el siguiente apartado se analiza la dispersión y repetibilidad de las medidas.

3.2.2. Estudio de repetibilidad

Una característica interesante de analizar de los sensores es la repetibilidad de las medidas, es decir, estudiar la variación entre medidas en diferentes instantes sin modificar el sistema.

Podemos observar que la medida leída por el sensor varía sin alterar el sistema. Para ello, debemos cuantificar la repetitividad o la varianza entre las medidas realizadas. Colocamos el objetivo a diferentes distancias (10 cm, 50 cm y 100 cm y ángulos (-5º, 0º y 5º) para obtener 10 muestras en diferentes instantes (con 1 segundo de diferencia) sin alterar la medida, y calculamos la varianza.

Tabla 3.1. Varianza de las medidas [cm]

	-5º	**0º**	**5º**
10 cm	0,055	0,029	0,015
50 cm	0,051	0,088	0,163
100 cm	0,121	0,055	0,159

Apreciamos que la varianza de lectura de las medidas es mínima y que solo en dos casos supera 1 mm, para 100 cm a- 5º y para 50 cm a 5º.
Además, realizamos 1000 medidas tomadas cada 1 segundo, en cinco puntos diferentes, con el objetivo de calcular la media, la varianza y la desviación estándar de la medida sin modificar el sistema.

Tabla 3.2. Resultados estudio de repetibilidad para 1000 muestras

Distancia [cm]	Ángulo [º]	Media [cm]	Varianza [cm]	Desviación [cm]
10	0	10,045	0,003	0,05
50	0	50,177	0,05	0,22
50	-10	49,899	0,04	0,21
50	10	50,069	0,03	0,19
100	0	99,764	0,05	0,24

Podemos apreciar que, aunque las medidas muestren cierta variación, la varianza máxima resulta ser muy pequeña, de menos de 0,5 mm, y la desviación estándar no alcanza los 3 mm en ninguno de los casos.

Ejemplo gráfico para el escenario con objetivo a 50 cm y 0º:

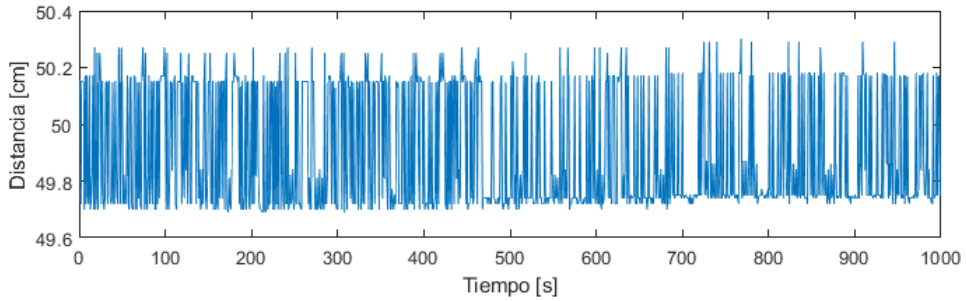

Fig. 3.7. 1000 muestras con objetivo a 50 cm y 0º.

Para entender mejor la figura anterior, realizamos el histograma de las medidas:

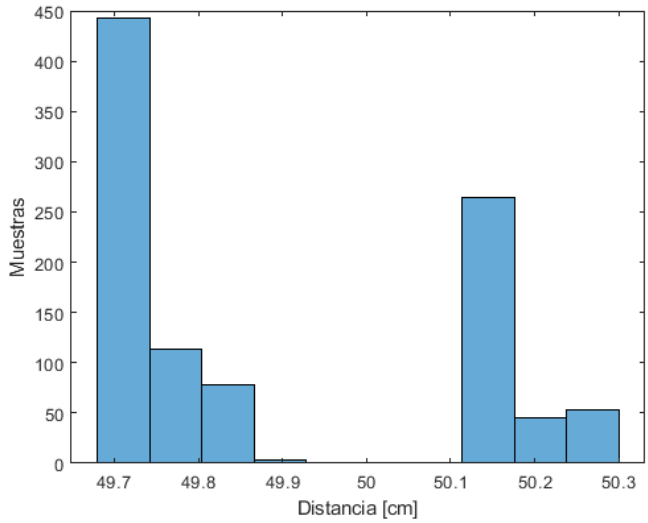

Fig. 3.8. Histograma para 1000 muestras con objetivo a 50 cm y 0º.

Podemos observar que no hay ninguna medida obtenida con valor exacto a 50,0 cm, pero que oscilan entre 49,7 cm y 50,3 cm. El error máximo de medida es de 3 mm, con una dispersión máxima de 6 mm. De esta manera, para obtener el

valor de la distancia al blanco, de la manera más precisa posible, deberíamos leer un grupo de muestras de distancias del sensor y realizar el promediado de las mismas.

3.2.3. Limitaciones del sensor

En definitiva, el sensor HC-SR04 debe operar con objetivos a una distancia mínima de 50 cm y dentro de un haz angular de máximo 10º por banda, para asegurar un error de medida inferior al 15% y estar por debajo de 1 cm. El alcance máximo obtenido, sin perder un objetivo de 15 cm de envergadura, es de alrededor de 2,5 m.

Por otro lado, el sensor muestra una precisión de repetición alta, de entre 0,5 mm y 2 mm en todos los casos estudiados (Tabla 3.1 y Tabla 3.2), de manera que la variabilidad de las medidas es muy pequeña.

En general, para ser un sensor de ultrasonidos sencillo, las lecturas obtenidas son muy aceptables en la zona operativa comentada. Aunque su punto débil es al haz angular, lo que podría acabar afectando al correcto funcionamiento del sistema de seguimiento.

CAPÍTULO 4. SISTEMA SONAR BIDIMENSIONAL

4.1. Principio de funcionamiento

El sonar bidimensional es un sistema que rastrea objetos en el plano horizontal XY, es decir que únicamente rota sobre el eje vertical Z, analizando azimut. Las dos aplicaciones principales son:

- Rastreo y detección de objetos en 360º:
  En este caso el sonar realiza giros a velocidad constante, analizando posibles objetivos, si detecta un objeto, muestra su posición y ángulo relativo en un gráfico polar. Un claro ejemplo de esta aplicación es un sonar submarino.

- Rastreo y seguimiento de objetos en 360º (tracker 2D):
  En este caso el sonar puede estar inactivo hasta que detecta un objeto, en el momento de la detección el sistema rastrea y persigue el objetivo angularmente. Un posible ejemplo de esta aplicación sería un seguidor solar fotovoltaico.

4.1.1. Configuración de los ultrasonidos

4.1.1.1. Configuraciones posibles

En el momento de configurar y disponer los sensores, hay varias posiciones posibles. Los sensores deben estar alineados en un mismo eje, pero es posible invertir uno de ellos o incluso mover el eje de referencia en el cálculo de las distancias tal y como se muestra en la siguiente figura, donde C representará el origen de coordenadas del sistema y A y B las posiciones puntuales que representarán a los sensores:

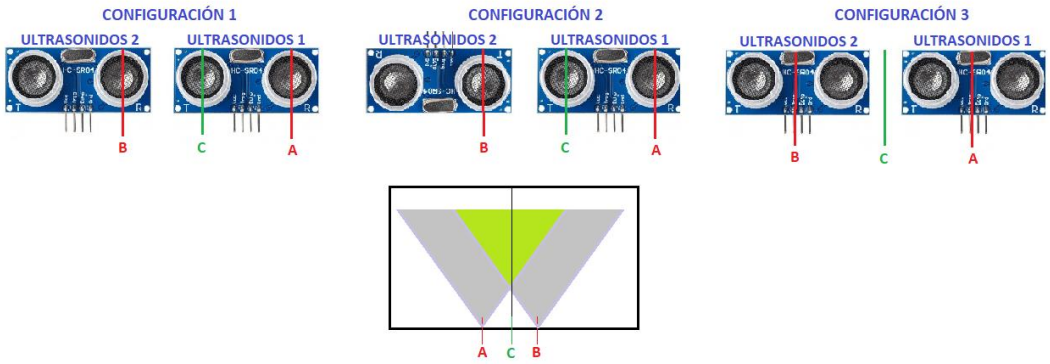

Fig. 4.1. Configuraciones de los sensores

Una vez elegida la configuración de los sensores, es necesario posicionarlos en un mapa de coordenadas. Para ello, fijamos los sensores a lo largo del eje X y los centramos en el eje Y, tal y como se muestra en la siguiente figura:

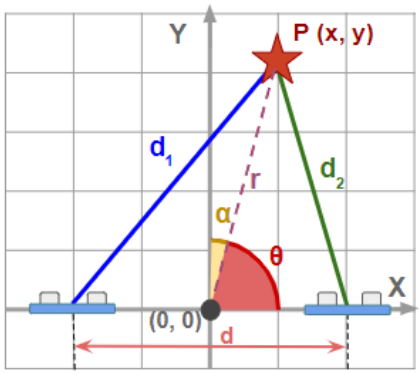

Fig. 4.2. Disposición de los sensores

De esta manera, para realizar el apuntamiento al objetivo, definimos el ángulo relativo al mismo α (azimut) sabiendo su posición polar (r, θ) donde: $\alpha = 90 - \theta$.

4.1.1.2. Área de trabajo útil

El seguidor bidimensional sólo será operativo en una cierta área. Esa zona operativa se define por la superposición de la cobertura de ambos sensores, es decir, cualquier objetivo que esté fuera de una de las zonas de cobertura de un único sensor no podrá ser rastreado ya que los algoritmos no funcionarían.

Para conocer el área de trabajo aproximada debemos calcular la distancia mínima a la que un objeto será detectado, esta distancia mínima dependerá de la distancia entre sensores (d) y del haz operativo de los mismos $(\Delta\theta)$:

$$d_{min} = \frac{d}{2\sin(\Delta\theta)} \qquad \textbf{(4.1)}$$

Por ejemplo, calculamos el área operativa separando los sensores 5 cm y la graficamos en la siguiente figura:

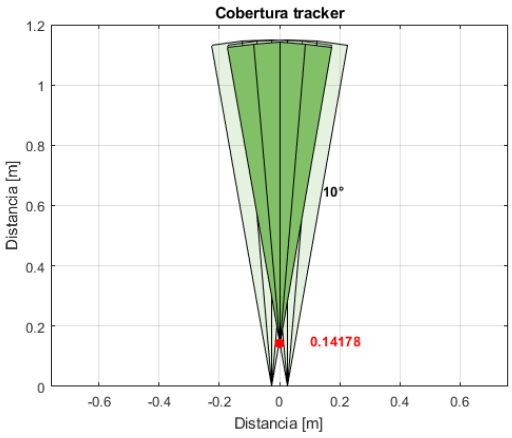

Fig. 4.3. Área operativa tracker 2D

Obteniendo una distancia mínima de detección de 14,2 cm tal y como se muestra con un punto rojo en la figura Fig. 4.3

.

4.2. Diseño

En este apartado analizamos el diseño del sonar bidimensional.

4.2.1. Primer prototipo

Para probar y verificar el sistema sonar bidimensional completamente, necesitamos crear un primer prototipo. A la hora de elegir el material de los componentes, las opciones posibles serían usar madera o plástico para facilitar el trabajo y asegurar un buen acabado estético.

El primer prototipo constará de un soporte para encajar los dos sensores donde irán encajados, un eje vertical y una base rotatoria que contendrá los elementos electrónicos.

4.2.1.1. Diseño en CAD

Para realizar el diseño del sistema definitivo utilizaremos software CAD, en este caso SolidWorks [15]. No es necesario profundizar en el uso y utilidades de SolidWorks y es posible seguir tutoriales de aprendizaje libre [16] para crear modelos y proyectos propios. El diseño del sonar bidimensional se muestra en la siguiente figura:

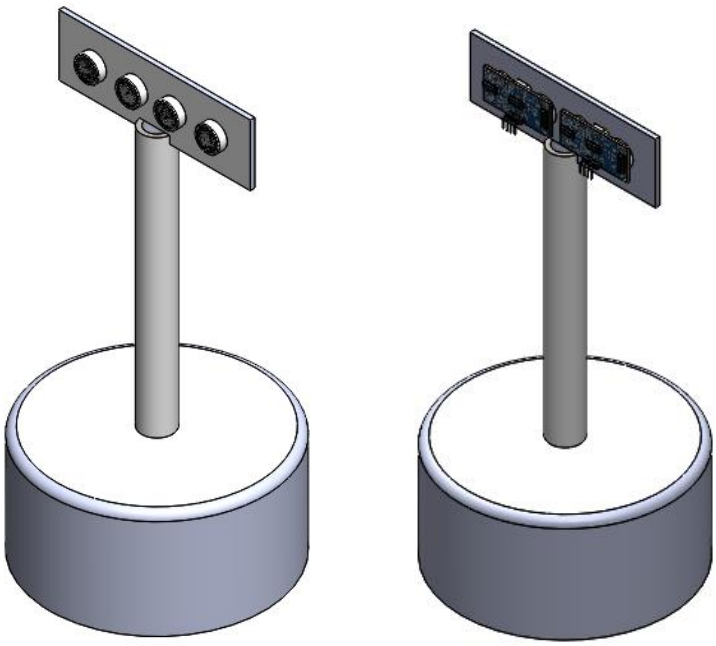

Fig. 4.4. Diseño CAD del tracker2D

El modelo del sonar bidimensional está disponible y puede ser revisado en el siguiente enlace: *http://bit.ly/recursos_TFG_sonar*.

En este caso, como el diseño no resulta muy complejo, realizamos un primer prototipo en plástico PVC (policloruro de vinilo), que se puede apreciar en las siguientes figuras:

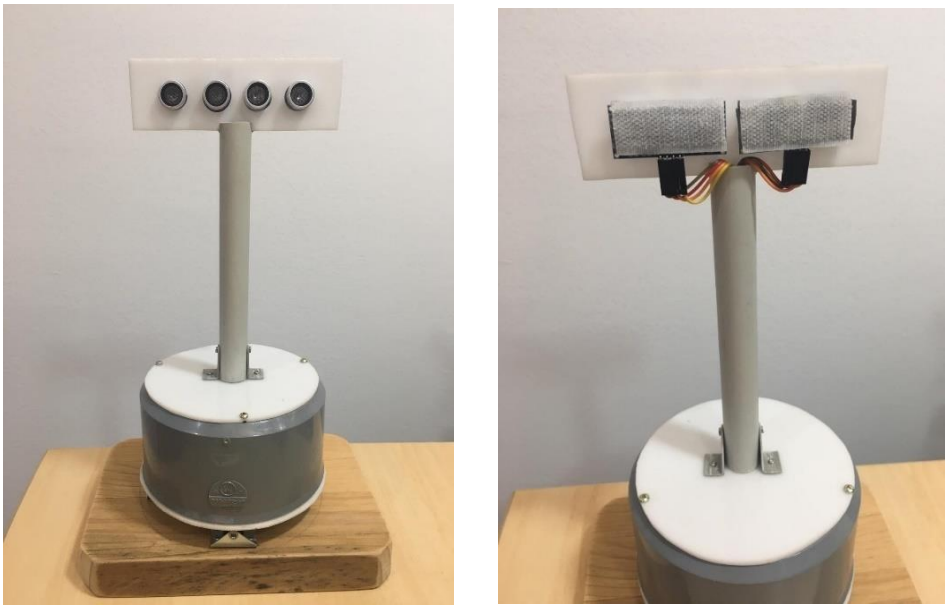

Fig. 4.5. a) Vista frontal del prototipo b) Vista trasera del prototipo

4.2.2. Ensayos

4.2.2.1. Configuraciones de los sensores

En este subapartado analizaremos las tres posibles opciones de disponer los sensores para decidir cuál de ellas ofrece mejores prestaciones.

Primeramente, compararemos la configuración 1 y 2 ya que la diferencia entre ambas es invertir uno de los sensores. Un primer análisis interesante sería comparar los errores relativos obtenidos usando estas disposiciones de sensores, para ello calculamos los diagramas de cobertura con error relativo:

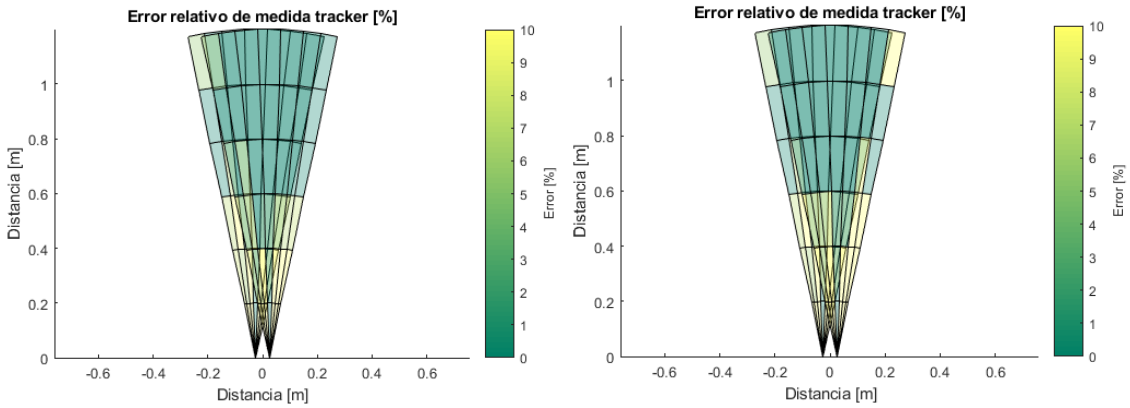

Fig. 4.6. a) Error relativo configuración 1 b) Error relativo configuración 2

Podemos apreciar que, en ambos casos, el error relativo es inferior al 5% en el área operativa superior a 50 cm y haz angular menor a 10º. Por tanto, el hecho de escoger la configuración 1 o 2 no afectaría a los resultados obtenidos y ambas serían válidas.

Para realizar el estudio, obtenemos datos en bruto (*raw data*) de las medidas obtenidas por los dos sensores, colocando el objetivo en múltiples puntos. Empezamos analizando las tres configuraciones posibles para disponer los sensores, usando el tercer algoritmo matemático para posicionar el objetivo.

Los resultados obtenidos se muestran la Tabla A.7 del apéndice y calculamos el error relativo de las medidas para realizar el análisis. Observando la Tabla A.8 extraemos las siguientes conclusiones:

- La configuración 1 y 2 muestran un error relativo elevado (de hasta un 21% en distancia y un 6% angular) para objetivos cercanos (< 50 cm). En cambio, para puntos más lejanos, el error disminuye notoriamente hasta el 1%.
- La configuración 3 muestra un mejor comportamiento en todos los puntos, comparando con las anteriores opciones. El error relativo en distancia máximo no alcanza el 10%, mientras que en ángulo es prácticamente despreciable.

Aunque para cortas distancias, uno de los sensores no alcanzaría el objetivo y podríamos estar fuera del área operativa de trabajo, se puede apreciar que los resultados de la configuración 3 son mejores que las demás opciones en regiones cercanas. En cambio, a mayores distancias parece ser que las tres opciones serían válidas ya que el error se reduce y se iguala. Esto indica que el eje de medida no está posicionado correctamente en las 2 primeras configuraciones y afecta a cortas distancias, pero que al alejarnos se reduce por el orden de magnitud de la distancia a la que se sitúa el objetivo en relación a la distancia entre los sensores. En conclusión, la mejor opción para trabajar sería la configuración 3.

4.2.2.2. Algoritmos matemáticos

Ahora es momento de estudiar los algoritmos matemáticos desarrollados para analizar los resultados y el comportamiento del sonar bidimensional.

Estudiamos los tres algoritmos matemáticos usando una misma disposición de los sensores. En este caso, realizaremos el análisis usando la configuración número 3, en la que consideramos como centro de coordenadas el punto medio entre los sensores.

Analizando los resultados obtenidos en la Tabla A.9 del apéndice calculamos el error relativo para cada posición del objetivo y realizamos la comparativa entre el resultado de los algoritmos. Del análisis de la Tabla A.10 del apéndice podemos apreciar que:

- El algoritmo 1 comete un error relativo en distancia de hasta un 21% y en ángulo de hasta un 14% para un objetivo a 10 cm. De hecho, ambos errores podrían considerarse elevados hasta que el objetivo se posiciona a un mínimo de 50 cm.
- El algoritmo 2 tiene un comportamiento parecido para distancias cercanas.
- El algoritmo 3 obtiene mejores resultados con el objetivo a corta distancia, comparando con los dos algoritmos anteriores. En este caso, podemos apreciar que el algoritmo, para distancias superiores a 50 cm comete un error inferior al 3%.

A cortas distancias los algoritmos podrían fallar por no cumplir la distancia mínima para estar dentro del área operativa de trabajo. Eso sí, podemos apreciar que el algoritmo 3 obtiene mejores resultados. De hecho, parece lógico ya que a distancias grandes la posición del objetivo podría trazarse con dos círculos debido al tamaño del radio en comparación a la distancia entre los sensores, pero que cortas distancias no sería posible y deberíamos recurrir al método de posicionado por elipses.

CAPÍTULO 5. SISTEMA SONAR TRIDIMENSIONAL

5.1. Principio de funcionamiento

El sonar tridimensional (tracker 3D) es un sistema que rastrea objetos, determinando su posición y girando hasta ella, en el plano horizontal XY en *azimut* (α) rotando sobre un eje vertical *z*, y en el plano vertical YZ, rotando en elevación (β) sobre un eje horizontal *x*.

El sonar debe constar de al menos dos ejes de rotación:

- Eje vertical (eje Z): para modificar el ángulo de azimut (α).
- Eje horizontal (eje X): para modificar el ángulo de elevación (β).

Para cada uno de los ejes debe haber un mecanismo adaptado para usar un motor paso a paso 28BYJ-48 que lo haga girar y un soporte para contener dos ultrasonidos HC-SR04.

En definitiva, el sistema debe determinar el ángulo en el que se encuentra el blanco respecto a un sistema de coordenadas y actuar sobre los motores para apuntar el tracker en la dirección del blanco.

Para facilitar el diseño y poder aprovechar toda la matemática realizada y estudiada los ultrasonidos se dispondrán de igual manera en ambos ejes. La distancia entre sensores puede variar, ya que es una constante que puede ser modificada y no afecta al uso de los algoritmos, pero se ha de respetar el posicionado usando una de las tres configuraciones posibles del apartado 4.1.1.

En la figura 5.1 podemos apreciar la disposición de los sensores seleccionada:

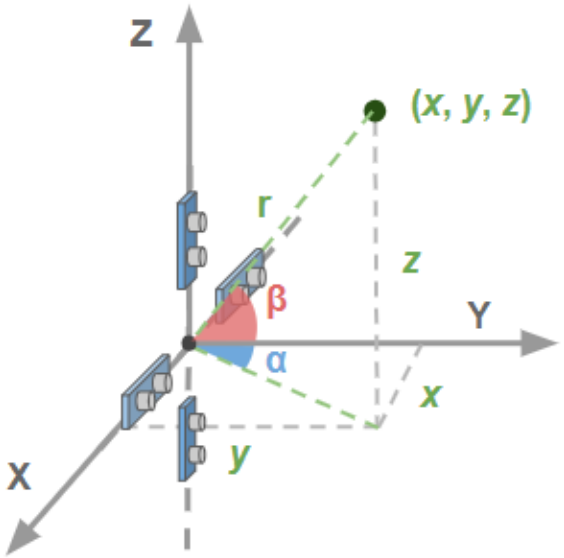

Fig. 5.1. Disposición de los sensores en tres dimensiones

5.2. Diseño en CAD

En este apartado se enumeran todos los componentes del sistema con el objetivo de caracterizarlos y proceder al diseño CAD.

5.2.1. Componentes del sistema

Como se ha comentado en el apartado anterior, necesitamos un soporte para encajar los cuatro sensores siguiendo la disposición de la siguiente figura:

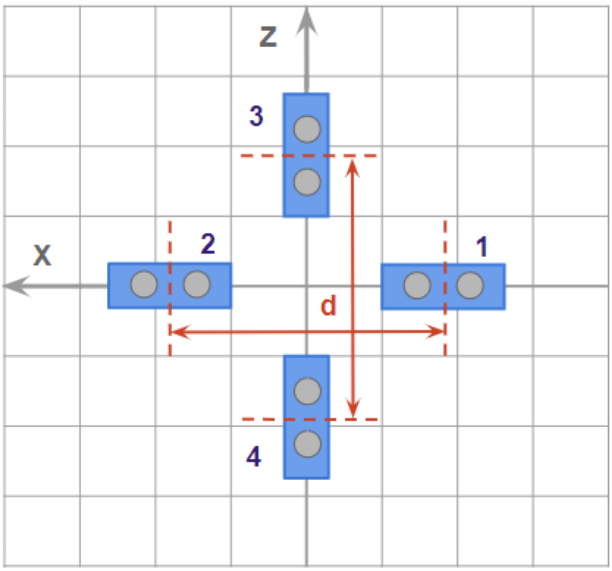

Fig. 5.2. Disposición de los sensores en el plano XZ

Conociendo las dimensiones del sensor HC-SR04:

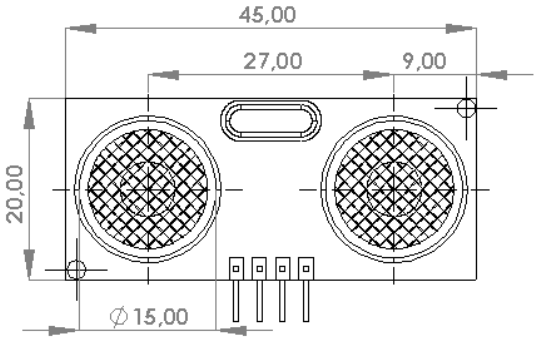

Fig. 5.3. Dimensiones del sensor HC-SR04, unidades en mm

Podemos deducir que la distancia de separación (d) mínima resulta ser de 65 mm:

$$d_{mín} = \frac{45}{2} + 20 + \frac{45}{2} = 65\ mm \qquad \textbf{(5.1)}$$

El soporte incluirá el eje horizontal, con su respectivo encaje para el eje de un motor, para permitir el seguimiento del objetivo en elevación.

Además de los orificios para los sensores de ultrasonidos, haremos un orificio para añadir un LED láser justo en el centro que permitirá comprobar la precisión de apuntamiento al objetivo.

El diseño escogido para definir el sistema de seguimiento tridimensional se muestra en la figura 5.4:

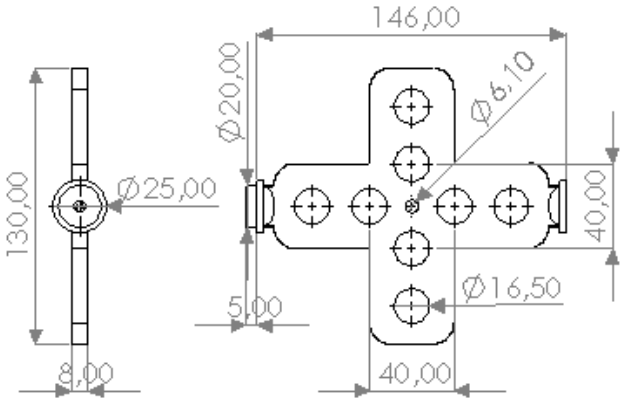

Fig. 5.4. Plano soporte vertical, unidades en mm

La idea principal es que el soporte vertical esté sujeto y rote sobre otra pieza. Como podemos apreciar en el plano anterior la pieza deberá tener un ancho interno de 146 mm, un grosor mínimo de 5 mm y dos orificios de diámetro (Φ=20 mm) para encajar el eje horizontal.

Además, esta pieza deberá permitir la sujeción de un motor para el eje de elevación y el encaje de un segundo motor para el eje de azimut. En el lado opuesto al motor del eje horizontal, se debe incluir un tope extraíble atornillado para encajar ambas piezas.

Las dimensiones del motor 28BYJ-48:

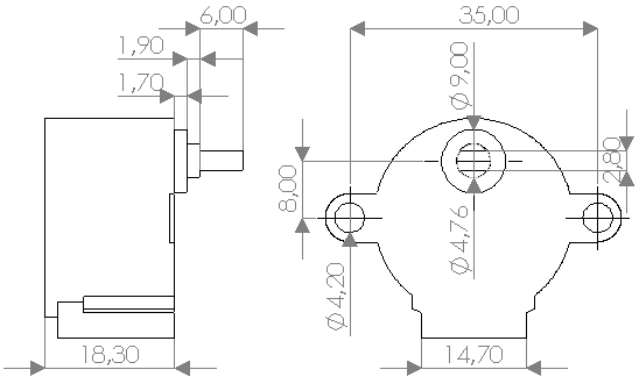

Fig. 5.5. Dimensiones motor 29BYJ-48, unidades en mm

Por un lado, observamos que el encaje para el eje debe ser un orificio de 8 mm de profundidad y 5 mm de diámetro con dos muescas laterales de 1 mm. Por otro lado, el soporte del motor constará de tres orificios, uno de diámetro 9 mm para pasar el eje y dos de diámetro 5 mm para fijarlo a la pieza.

Así que procedemos a dibujar el soporte horizontal:

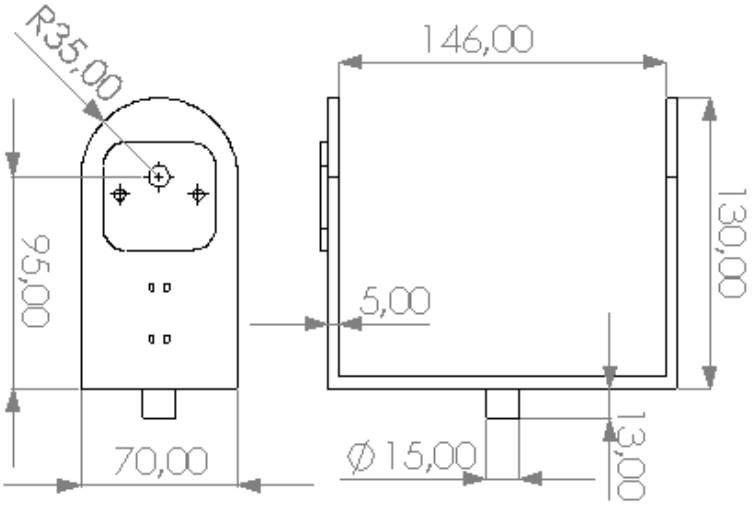

Fig. 5.6. Plano soporte horizontal, unidades en mm

Finalmente, necesitamos una base para encajar el eje vertical y contener parte de los componentes electrónicos. La base constará de un rodamiento sobre el cual reposará y rotará todo el sistema, permitiendo contrarrestar el peso y realizar movimientos finos.

Realmente podríamos utilizar como base cualquier caja, pero es necesario diseñar un último soporte para contener el rodamiento y fijar el segundo motor. En este caso, usamos el rodamiento 6002-2RS de SMB [17] de diámetro interior 15 mm, exterior 32 mm y altura de 9 mm.

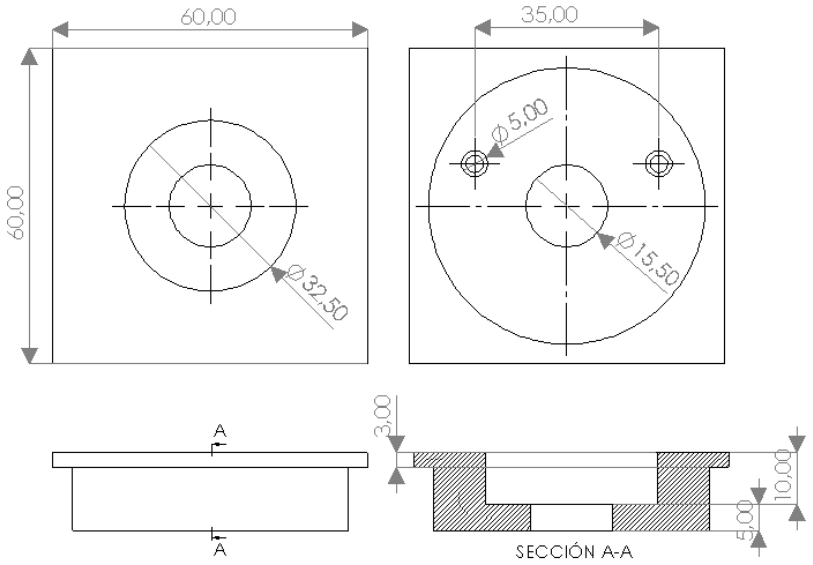

Fig. 5.7. Diseño soporte base, unidades en mm

5.2.2. Sistema completo

Una vez tenemos todas las piezas necesarias diseñadas y creadas en CAD, procedemos a realizar el ensamblaje del sistema completo. Para ello, incluimos todos los soportes explicados en el apartado anterior, el rodamiento y los todos componentes electrónicos. Los modelos CAD de los sensores, motores y de la placa Arduino están disponibles y pueden descargarse en GrabCAD [18].

Seguidamente, mostramos el ensamblaje del sistema completo:

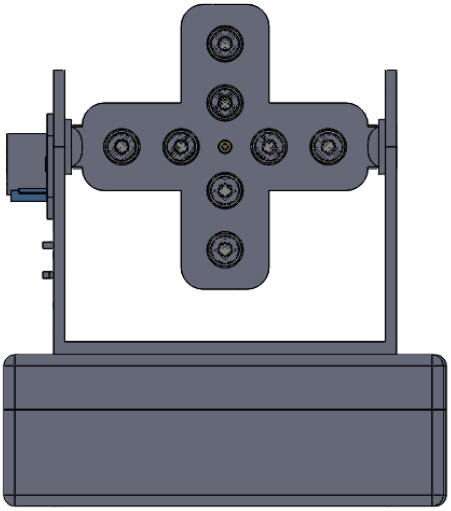

Fig. 5.8. Vista frontal del sonar tridimensional

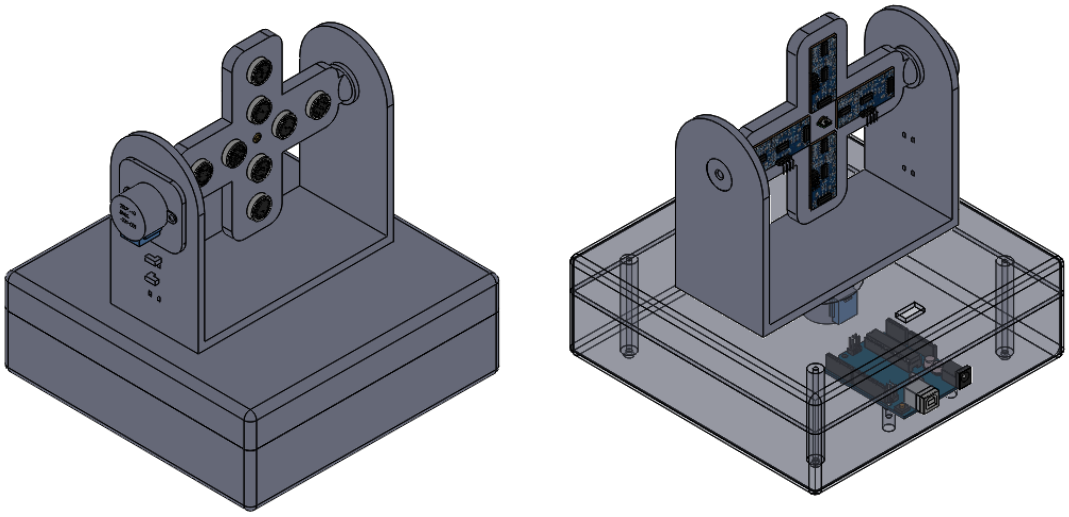

Fig. 5.9. Vistas isométrica frontal y posterior (con transparencia) del sonar

5.2.3. Ensamblaje

Primero de todo, para realizar el ensamblaje y montaje del sistema, debemos obtener todas las piezas y componentes listados. Las piezas diseñadas en CAD se han imprimido en material plástico PETG, con un 20% de relleno y grosor de capa de 100 µm en 3DHubs [19].

Para fijar los componentes sería necesario utilizar tornillos, pero considerando la fragilidad del material no atornillaremos directamente sobre el plástico. Para ello utilizaremos insertos de latón roscados de métrica 3.

Fig. 5.10. Insertos de latón roscados [20]

Por ejemplo, en este caso usamos tornillos de métrica 3 y profundidad de 5 mm, por lo que el orificio a introducir el inserto de latón debe ser de diámetro 5 mm y profundidad de 6 mm. Para encajarlo en el orificio, debemos colocarlo en el mismo y aplicar calor con un soldador sin aplicar fuerza, de esta manera el inserto penetrará el plástico y quedará totalmente fijado.

Para ensamblar el seguidor, podríamos empezar realizando un orificio de diámetro 16 mm en la tapa superior de la caja a utilizar. Seguidamente colocamos el rodamiento en el soporte para la base y fijándolo en el interior de la caja con adhesivo, es importante que los orificios de la caja y del soporte queden colineales para poder ajustar el eje vertical perfectamente.

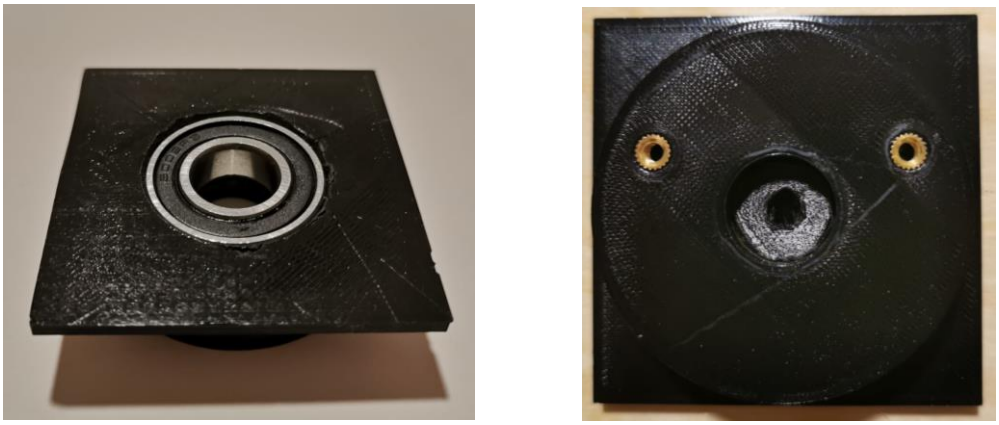

Fig. 5.11. a) Soporte base con rodamiento b) Soporte base fijado a la caja

Una vez fijado el soporte de la base, podemos colocar la pieza horizontal, insertando el eje en el interior del rodamiento y encajándolo en el primer motor. El primer motor debe atornillarse al soporte de la base en el interior de la caja.

Una vez colocada la pieza horizontal, procedemos a fijar el segundo motor en la pieza horizontal:

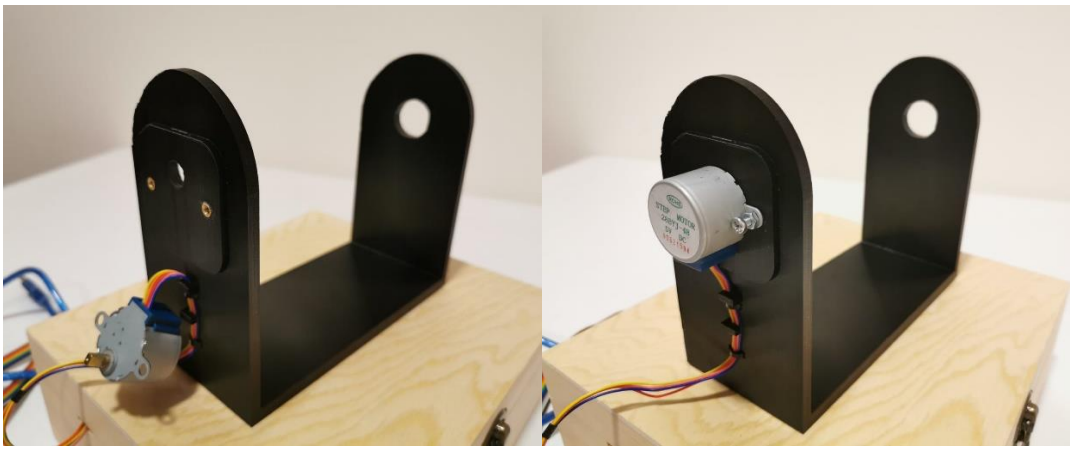

Fig. 5.12. Fijación del motor 2 a la pieza horizontal

Seguidamente, en el soporte vertical fijamos los sensores de ultrasonidos y el LED láser en sus respectivos orificios. El soporte vertical debe ajustarse en la pieza horizontal asegurando encajar el eje del motor en el mismo.

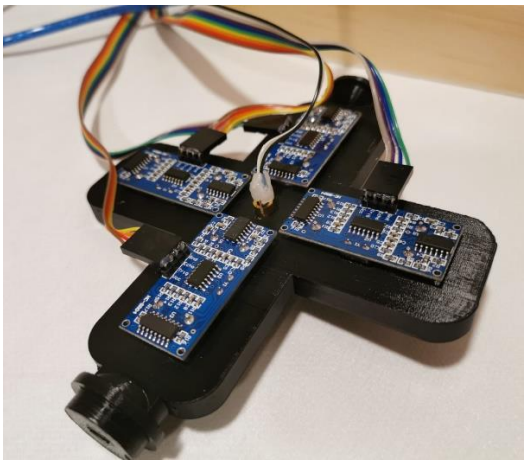

Fig. 5.13. a) Soporte horizontal b) Fijación del eje del motor

Finalmente, para fijar el eje horizontal usamos el tope diseñado, colocándolo en el orificio y atornillándolo al soporte vertical.

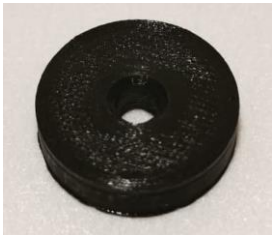

Fig. 5.14. Pieza de fijación del eje horizontal

Una vez realizados todos los pasos anteriores debemos realizar el cableado y conexionado de todos los sensores, motores y del LED a la placa de Arduino que se encuentra en el interior de la caja.

El resultado final es el siguiente:

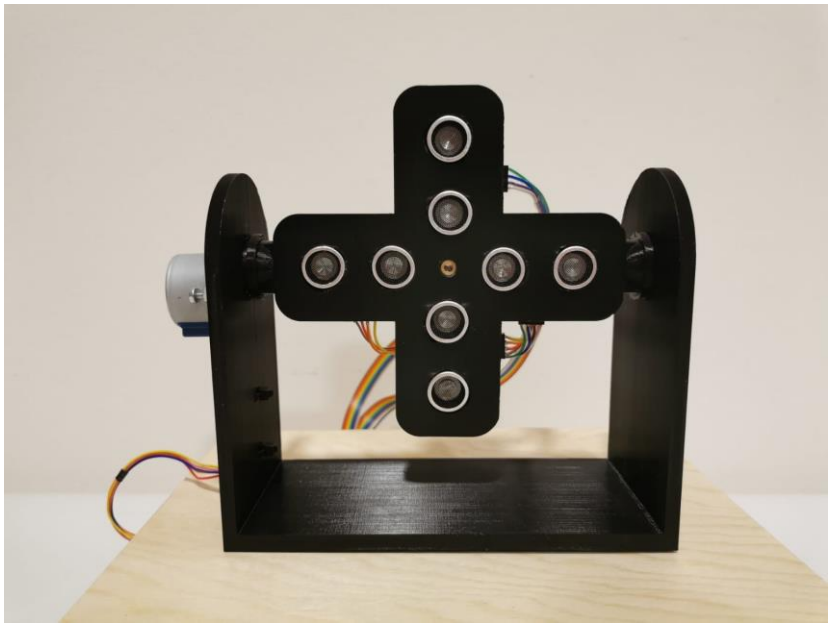

Fig. 5.15. Sistema sonar tridimensional

Y el interior de la caja:

Fig. 5.16. Interior de la caja del sistema

5.3. Ensayos

Para caracterizar el comportamiento del seguidor tridimensional se han realizado diferentes pruebas con objeto de obtener el error cometido y la precisión de apuntamiento a cierto blanco.

Considerando los resultados del seguidor bidimensional, se utilizarán la configuración y el algoritmo matemático número 3 para calcular la posición del objetivo.

5.3.1. Funcionamiento

Para empezar, se realiza el montaje de un escenario controlado, usando la mesa calibrada, para estudiar el funcionamiento del sistema colocando un objetivo en diferentes puntos. El objetivo es una caja metálica con un soporte de poliexpán (baja reflexión acústica) para poder fijarlo a diferente altura y no alterar la medida.

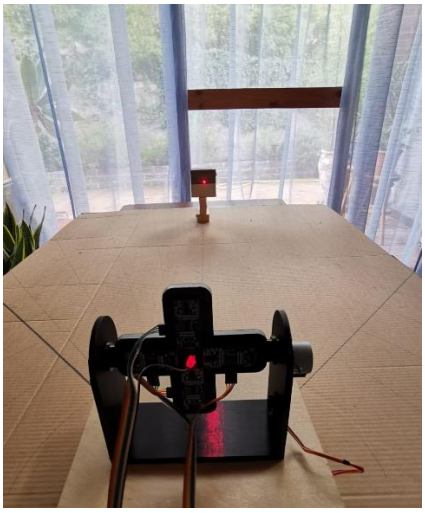

Fig. 5.17. Montaje para realizar el estudio

En este caso, el estudio es estático con motores desactivados y realizando 1000 medidas por punto. El escenario consta de 4 posiciones tal y como se muestra en la siguiente figura:

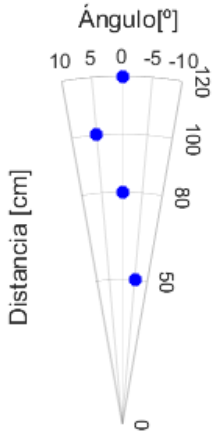

Fig. 5.18. Escenario de estudio

Para llevar a cabo las medidas y procesarlas usamos el código adjunto en el apéndice B. Separamos el problema en dos partes y primero realizamos el estudio en el plano horizontal. En la siguiente tabla mostramos la posición real del objetivo (según el escenario anterior), la media de las 1000 lecturas de los 4 sensores de ultrasonidos y la posición obtenida por el algoritmo:

Tabla 5.1. Resultado obtenido en plano horizontal

Posición real del objetivo		Lectura de los sensores [cm]				Posición calculada	
Distancia [cm]	Ángulo relativo [°]	Distancia 1	Distancia 2	Distancia 3	Distancia 4	Distancia [cm]	Azimut [°]
50	-5	48,57	51,21	49,67	50,12	49,86	-5,235
80	0	80,21	79,87	80,72	80,55	79,58	0,885
120	0	119,23	120,83	119,75	120,15	120,37	-0,2795
100	5	100,75	99,7	99,67	99,82	99,67	5,135

Y calculamos el error absoluto cometido en estas posiciones. Observando la Tabla A.11 del apéndice, el error cometido por el sistema es muy pequeño, siendo inferior a 4 mm en distancia y 1º en ángulo para los cuatro casos. Realizamos un procedimiento similar en el plano vertical, fijando el objetivo a 50 cm de distancia, pero modificando el ángulo de elevación

Tabla 5.2. Resultado obtenido en plano vertical

Posición real del objetivo		Lectura de los sensores [cm]				Posición calculada	
Distancia [cm]	Ángulo relativo [°]	Distancia 1	Distancia 2	Distancia 3	Distancia 4	Distancia [cm]	Elevación [°]
50	-5	49,78	50,12	49,17	50,78	51,21	-5,58
50	0	49,73	50,23	49,77	49,95	49,86	-1,27
50	5	49,67	50,52	51,15	49,36	48,12	6,21

Volvemos a calcular el error absoluto para este caso. En la Tabla A.12, apreciamos que en el plano vertical el error obtenido es superior, alcanzando 1 cm en distancia y 1º en ángulo, aunque cabe destacar que el posicionado real del objetivo en este caso no es totalmente exacto.

Como disponemos de 1000 muestras de medidas por cada una de las posiciones estudiadas, realizamos histogramas para analizar los resultados con más detalle.

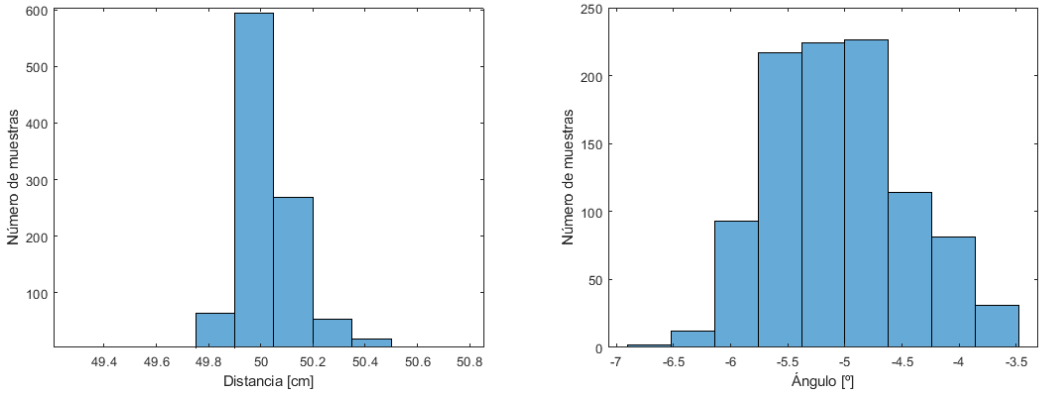

Fig. 5.19. a) Distancia para 50 cm y -5º **b)** Ángulo para 50 cm y -5º

En este primer caso, observamos una distribución gaussiana tanto en las medidas de distancia como en las de ángulo. En el caso de la distancia,

apreciamos que la media es cercana a 50 cm y hay una dispersión máxima de 4 mm, para el ángulo, el valor medio ronda los -5º y hay casos con un error angular máximo de 1,5º.

En otra posición del objetivo posible, a 120 cm y 0º, el resultado es muy parecido:

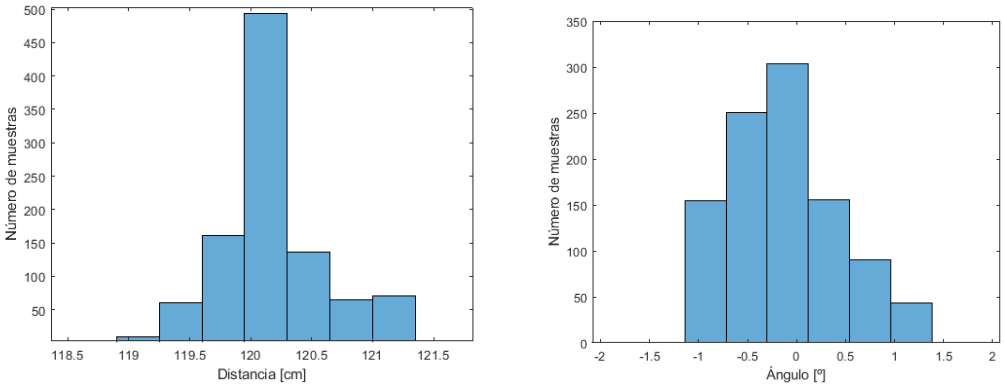

Fig. 5.20. a) Distancia para 120 cm y 0º **b)** Ángulo para 120 cm y 0º

En un último caso, pero a 100 cm y 5º, también:

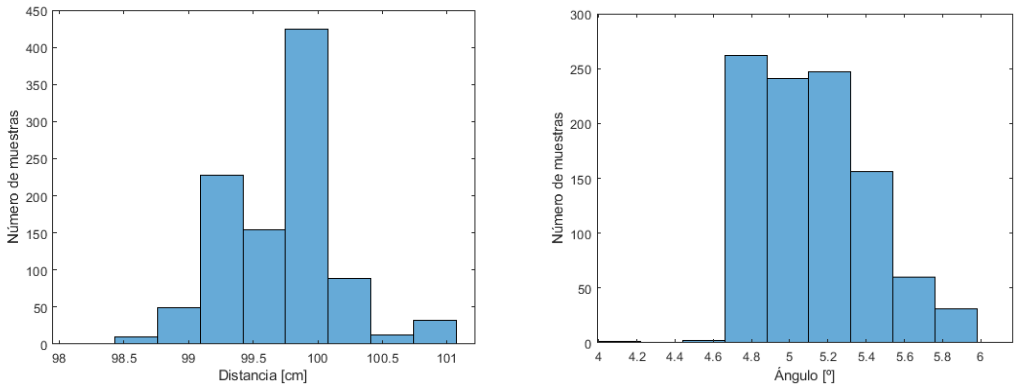

Fig. 5.21. a) Distancia para 100 cm y 5º **b)** Ángulo para 100 cm y 5º

En resumen, apreciamos que las medidas tienen un valor medio cercano al real y al esperado, tanto en distancia como en ángulo, y que el error máximo en distancia es alrededor de 1 cm y en ángulo de 1,5º. Resulta destacable la variabilidad de las medidas, sin alterar el escenario, tal y como hemos comentado y estudiado en el apartado 3.2.2. Esta variabilidad de medida de los sensores provoca una gran sensibilidad angular en el sistema, que simula una corrección constante de la posición del objetivo, aunque no se mueva.

5.3.2. Limitaciones del sistema

Después de realizar los ensayos, observar el comportamiento del sistema y analizar los resultados obtenidos, apreciamos que el sistema tiene ciertas limitaciones que afectan al correcto funcionamiento del mismo.

La precisión en distancia de los sensores de ultrasonidos es bastante correcta (por debajo de 1 cm) pero tienen un haz angular de trabajo muy reducido (inferior a 10º de cobertura) que facilita perder el objetivo fácilmente, ya que salimos del

área operativa del sistema. Para evitarlo, debemos asegurar que los cuatro sensores vean al objetivo y no realizar movimientos rápidos y bruscos.

Por otro lado, como acabamos de comentar, el sistema presenta una variabilidad en las medidas que puede generar correcciones innecesarias de la posición del objetivo y provocar que algún sensor pierda el blanco.

El objetivo a seguir debe respetar unas dimensiones mínimas, para posiciones cercanas basta con un objeto de una envergadura de unos 5 cm, pero para distancias superiores a 70 cm el objetivo debería ser al menos de 15 cm. Los resultados mejoran si el objetivo es de material sólido y metálico. El sistema deja de responder correctamente para distancias superiores a 2,5 m ya que los sensores dejan de ver cualquier objetivo con precisión.

Además, por motivos físicos, el sistema no puede realizar movimientos completos en todas las direcciones ya que el cableado los limita.

CAPÍTULO 6. CONCLUSIONES

6.1. Resultados

Observando los resultados obtenidos durante el proyecto, podemos afirmar que se ha logrado el objetivo de desarrollar un sistema sonar de seguimiento. El sistema obtenido realiza el seguimiento de un blanco siempre que se respeten las limitaciones comentadas en el apartado anterior.

Hay que considerar que el sistema puede ser mejorado, estudiando otras alternativas posibles más complejas y caras, que no cumplirían el objetivo económico y didáctico del proyecto.

6.2. Mejoras futuras

En este apartado se enumeran algunas de las opciones posibles para mejorar el funcionamiento y los resultados obtenidos del sistema.

6.2.1.1. Mejoras en los sensores

Sería posible añadir más sensores en ambos ejes con el objetivo de ampliar el área de cobertura de trabajo y mejorar la robustez del sistema. Aunque aumentaría notoriamente la complejidad del diseño, de los algoritmos matemáticos y del software de control.
Además de probar diferentes configuraciones de los sensores, por ejemplo, en lugar de alinearlos en un eje, añadir cierto ángulo entre ellos como se muestra en la siguiente figura:

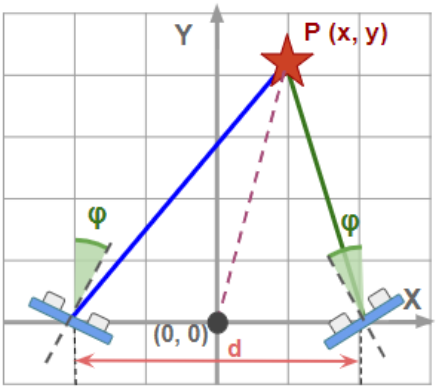

Fig. 6.1. Alternativa con sensores no alineados

6.2.1.2. Motores alternativos

Una buena opción para mejorar el funcionamiento de los motores sería usar motores de continua con codificadores, explicados en el apartado 2.2.3.1. Estos motores ofrecen una precisión angular muy buena y permiten comprobar el ángulo real en el cual se posicionan, aunque son complejos y caros.

Al tener codificador, es posible definir un sistema retroalimentado que compara el ángulo ordenado con el realizado, permitiendo correcciones simultaneas del mismo. Es decir, podemos implementar un controlador PID (proporcional, integral y derivativo) para evaluar el error entre el ángulo medido y el deseado.

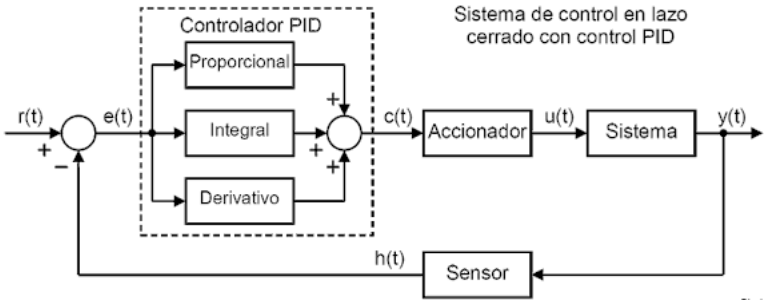

Fig. 6.2. Esquema controlador PID [21]

Definiendo la señal error e(t) como la diferencia entre la respuesta deseada y la respuesta obtenida por el motor; el PID se caracteriza con tres parámetros de control: K_p, K_i y $K_d$que varían los pesos de la ecuación del controlador:

$$c(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \cdot \frac{\partial e(t)}{\partial t} \qquad \textbf{(6.1)}$$

De esta manera, el controlador PID evaluaría el error realizado por el motor y lo corregiría aplicando la ecuación anterior. Los valores de los parámetros de control son totalmente sintonizables y se fijan según la velocidad de respuesta y la estabilidad del sistema deseadas, y el error máximo tolerado por el usuario.

6.3. Presupuesto

En este apartado listamos todos los componentes y costes del proyecto para realizar el sistema sonar de seguimiento tridimensional.

Tabla 6.6.1. Presupuesto del sonar 3D

Tipo	Modelo	Subministrador	Cantidad	Precio unitario	Precio
Sensor ultrasonidos	HC-SR04	AZ-Delivery	4	1,75 €	6,99 €
Motor paso a paso	28BYJ-48	Iberobotics	2	2,44 €	4,88 €
LED láser	B0716FSRJ1	AZ-Delivery	1	0,71 €	0,71 €
Placa arduino	Arduino UNO R3	AZ-Delivery	1	7,99 €	7,99 €
Cableado	Flat Cable 1,27	Aussel	1	0,95 €	0,95 €
Conectores	Kit dupont	Arceli	1	1,00 €	1,00 €
Caja de madera	Caja 20x20x15 cm	Zeller	1	4,50 €	4,50 €
Inserto de latón	Kit tuercas M2-M5	Yanshon	1	1,30 €	1,30 €
Pieza 3D	Móbil horizontal	3DHubs	1	52,07 €	52,07 €
Pieza 3D	Móbil vertical	3DHubs	1	29,10 €	29,10 €
Pieza 3D	Soporte base	3DHubs	1	16,22 €	16,22 €
				Total	**125,72 €**

CAPÍTULO 7. BIBLIOGRAFIA

[1] © Arduino, “Arduino - Home,” 2019. [Online]. Available: https://www.arduino.cc/. [Accessed: 15-Sep-2019].

[2] “Sensor ultrasónico, mideindo la distancia en un sistema de aparcamiento - Electrogeek.” [Online]. Available: https://www.electrogeekshop.com/sensor-ultrasonico-mideindo-la-distancia-en-un-sistema-de-aparcamiento/. [Accessed: 28-Sep-2019].

[3] © Arduino, “Arduino - Products,” 2019. [Online]. Available: https://www.arduino.cc/en/Main/Products. [Accessed: 15-Sep-2019].

[4] © 2019 SICK AG, “© 2019 SICK AG.” [Online]. Available: https://www.sick.com/es/es/sensores-de-distancia/sensores-de-ultrasonido/um12/c/g305961. [Accessed: 15-Sep-2019].

[5] “Medir distancia con Arduino y sensor de ultrasonidos HC-SR04.” [Online]. Available: https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/. [Accessed: 28-Sep-2019].

[6] “Sensor ultrasonidos HC-SR04 - infootec.net.” [Online]. Available: https://www.infootec.net/sensor-ultrasonidos-hc-sr04/. [Accessed: 28-Sep-2019].

[7] Actobotics - ServoCity®, “Premium Planetary Gear Motors with Encoders.” [Online]. Available: https://www.servocity.com/motors-actuators/gear-motors/standard-duty-gear-motors/premium-planetary-gear-motors-with-encoders. [Accessed: 15-Sep-2019].

[8] MCI - Electronics, “Motores paso a paso ‘stepper’ - MCI Capacitación,” 2019. [Online]. Available: http://cursos.mcielectronics.cl/2019/06/18/motores-paso-a-paso-stepper/. [Accessed: 15-Sep-2019].

[9] © Prometec, “Motores paso a paso: 28BYJ-48 | Tienda y Tutoriales Arduino,” *Motores paso a paso: stepper 28BYJ-48*, 2017. [Online]. Available: https://www.prometec.net/motor-28byj-48/. [Accessed: 15-Sep-2019].

[10] Hitec - ServoCity®, “Hitec Servos,” 2019. [Online]. Available: https://www.servocity.com/servos/hitec-servos. [Accessed: 15-Sep-2019].

[11] Nema series - Electromate®, “NEMA Frame Stepper Motors - Explosion Proof Stepper Motors,” 2019. [Online]. Available: https://www.electromate.com/products/stepper-motors/nema-explosion-proof-stepper-motors/#cds-attribute-units. [Accessed: 15-Sep-2019].

[12] © Arduino, “Arduino - Software,” 2019. [Online]. Available: https://www.arduino.cc/en/main/software. [Accessed: 15-Sep-2019].

[13] Inc. © The MathWorks, “MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink,” *2019*, 2019. [Online]. Available: https://la.mathworks.com/. [Accessed: 20-Sep-2019].

[14] Visual Code - © Microsoft, “Visual Studio Code - Code Editing. Redefined,” 2019. [Online]. Available: https://code.visualstudio.com/. [Accessed: 20-Sep-2019].

[15] “3D CAD Design Software | SOLIDWORKS.” [Online]. Available: https://www.solidworks.com/. [Accessed: 20-Oct-2019].

[16] © SolidWorks Corporation, “SOLIDWORKS Tutorials | Resource Center | SOLIDWORKS,” 2019. [Online]. Available: https://www.solidworks.com/sw/resources/solidworks-tutorials.htm. [Accessed: 21-Sep-2019].

[17] S. Bearings, “6002-2RS sealed bearing 15x32x9mm (15mm x 32mm x 9mm).”

[18] © GrabCAD, “Popular models | 3D CAD Model Collection | GrabCAD Community Library,” *2019*, 2019. [Online]. Available: https://grabcad.com/library. [Accessed: 22-Sep-2019].

[19] © 3D Hubs, “3D Hubs | On-demand Manufacturing: Quotes in Seconds, Parts in Days,” 2019. [Online]. Available: https://www.3dhubs.com/. [Accessed: 22-Sep-2019].

[20] © Shoptronica S.L., “Insertos Roscados Latón 1036 y MDF,” 2019. [Online]. Available: https://www.shoptronica.com/tornilleria-especial-tuercas/5211-insertos-roscados-laton-1036-para-plastico-y-mdf-8944748451781.html. [Accessed: 22-Sep-2019].

[21] G. S. Picuino, “Controladores PID - Picuino.” [Online]. Available: https://sites.google.com/site/picuino/pid_controller. [Accessed: 22-Sep-2019].

APÉNDICE A. Tablas de medidas y resultados

Tabla A.1. Medidas sensor para cuadrante izquierda (ángulos negativos)

Ángulo [°]	Distancia [cm]	Medida [cm]										Distancia [cm]	Error [cm]	Error [%]
0	5	5,05	4,99	5,09	5,01	5,09	4,99	5,00	5,04	4,99	4,99	5,03	0,03	0,51%
0	10	10,08	9,83	10,14	9,72	10,07	9,72	9,83	10,15	9,72	10,14	9,94	0,06	0,60%
0	30	29,72	30,12	30,57	30,26	30,17	30,25	29,83	30,12	30,20	29,92	30,12	0,12	0,39%
0	50	50,87	49,85	50,85	49,87	50,12	50,47	50,38	50,52	49,85	50,30	50,31	0,31	0,62%
0	70	70,15	71,20	69,87	69,85	70,26	70,30	70,15	71,20	70,58	70,87	70,44	0,44	0,63%
0	100	99,92	100,04	99,08	99,18	99,51	99,18	99,08	99,18	99,06	99,18	99,34	0,66	0,66%
-5	5	6,43	5,97	5,18	5,87	5,87	5,97	5,84	5,97	6,84	6,74	6,07	1,07	21,36%
-5	10	11,25	10,96	11,06	10,96	11,06	10,78	11,06	10,96	11,06	10,96	11,01	1,01	10,11%
-5	30	30,21	31,30	30,64	31,63	31,64	31,20	30,52	31,63	31,20	31,32	31,13	1,13	3,76%
-5	50	50,72	50,91	51,27	50,09	50,72	50,77	53,78	50,34	50,89	51,32	51,08	1,08	2,16%
-5	70	70,58	71,63	70,30	70,28	70,69	70,73	70,58	71,63	71,01	71,30	70,87	0,87	1,25%
-5	100	99,58	99,56	99,58	100,11	99,58	100,54	99,15	99,66	99,99	99,56	99,73	0,27	0,27%
-7,5	5	6,12	6,02	5,92	5,92	5,92	6,02	6,02	6,02	6,02	6,02	6,00	1,00	20,00%
-7,5	10	10,58	10,58	10,70	10,60	10,68	10,60	10,70	10,60	10,60	10,70	10,63	0,63	6,34%
-7,5	30	33,12	33,02	33,46	33,02	33,02	33,02	33,14	33,02	33,02	33,03	33,09	3,09	10,29%
-7,5	50	54,20	53,77	52,93	53,36	54,20	53,66	53,75	52,00	52,89	54,20	53,50	3,50	6,99%
-7,5	70	70,85	72,16	70,57	70,55	70,96	71,00	70,85	71,90	71,28	71,57	71,17	1,17	1,68%
-7,5	100	99,66	99,56	99,66	99,56	99,68	99,99	99,66	99,15	100,08	99,13	99,61	0,39	0,39%
-10	5	5,92	5,92	5,81	5,81	5,81	6,38	6,38	5,92	6,38	5,92	6,03	1,03	20,50%
-10	10	10,99	10,60	11,11	11,01	10,70	10,60	10,68	10,58	11,01	10,70	10,80	0,80	7,98%
-10	30	32,05	32,05	32,17	32,48	32,04	32,14	32,48	32,07	32,17	32,07	32,17	2,17	7,24%
-10	50	52,05	52,07	52,50	52,05	52,07	52,50	52,50	52,05	52,50	52,50	52,28	2,28	4,56%
-10	70	71,44	74,74	72,79	73,14	73,89	72,59	73,44	73,90	73,96	72,86	73,27	3,27	4,68%
-10	100	108,86	108,86	108,87	108,77	102,48	108,84	108,41	81,67	109,29	108,84	105,49	5,49	5,49%

Tabla A.2. Medidas sensor para cuadrante derecho (ángulos positivos)

Ángulo [°]	Distancia [cm]	Medida [cm]										Distancia [cm]	Error [cm]	Error [%]
0	5	5,05	4,99	5,09	5,01	5,09	4,99	5,00	5,04	4,99	4,99	5,03	0,03	0,51%
0	10	10,08	9,83	10,14	9,72	10,07	9,72	9,83	10,15	9,72	10,14	9,94	0,06	0,60%
0	30	29,72	30,12	30,57	30,26	30,17	30,25	29,83	30,12	30,20	29,92	30,12	0,12	0,39%
0	50	50,87	49,85	50,85	49,87	50,12	50,47	50,38	50,52	49,85	50,30	50,31	0,31	0,62%
0	70	70,15	71,20	69,87	69,85	70,26	70,30	70,15	71,20	70,58	70,87	70,44	0,44	0,63%
0	100	99,92	100,04	99,08	99,18	99,51	99,18	99,08	99,18	99,06	99,18	99,34	0,66	0,66%
5	5	6,89	6,79	6,79	6,79	6,89	6,91	6,89	6,81	6,79	6,79	6,83	1,83	36,68%
5	10	11,11	11,11	11,11	11,11	11,23	11,11	10,82	11,11	11,22	11,11	11,10	1,10	11,04%
5	30	31,42	31,54	31,42	31,42	31,44	31,42	31,42	31,42	31,87	31,97	31,53	1,53	5,11%
5	50	50,94	50,92	50,92	51,78	51,78	51,33	51,04	50,92	50,92	50,89	51,14	1,14	2,29%
5	70	70,75	71,80	70,47	70,45	70,86	70,90	70,75	71,80	71,18	71,47	71,04	1,04	1,49%
5	100	101,02	100,68	100,59	100,56	100,16	101,00	100,69	100,14	100,16	101,10	100,61	0,61	0,61%
7,5	5	7,25	7,15	7,15	7,27	6,40	6,79	6,38	6,28	6,28	7,13	6,81	1,81	36,16%
7,5	10	12,30	12,28	12,40	12,30	12,40	12,28	11,99	12,28	12,40	11,42	12,21	2,21	22,05%
7,5	30	32,02	32,48	32,05	32,02	32,16	32,04	32,02	32,14	32,02	32,02	32,10	2,10	6,99%
7,5	50	51,85	52,28	51,83	51,83	52,26	52,28	51,85	52,69	51,83	52,28	52,10	2,10	4,20%
7,5	70	71,15	72,20	70,87	70,85	71,26	71,30	71,15	72,20	71,58	71,87	71,44	1,44	2,06%
7,5	100	100,69	101,27	100,67	101,17	99,89	100,68	100,95	99,83	100,68	100,68	100,65	0,65	0,65%
10	5	6,38	6,28	6,28	6,28	6,38	6,40	6,38	6,38	6,38	6,28	6,34	1,34	26,84%
10	10	11,99	12,04	11,97	12,09	11,99	12,09	11,97	12,09	11,99	12,09	12,03	2,03	20,31%
10	30	32,09	32,19	32,07	32,09	32,64	32,54	32,09	32,62	32,09	32,11	32,25	2,25	7,51%
10	50	52,87	52,28	51,87	51,65	52,63	52,28	51,05	51,85	51,83	52,17	52,05	2,05	4,10%
10	70	70,48	71,53	70,20	70,18	70,59	70,63	70,48	71,53	70,91	71,20	70,78	0,78	1,11%
10	100	100,08	100,20	100,52	100,09	98,74	100,09	100,97	90,97	100,11	100,23	99,20	0,80	0,80%

Tabla A.3. Error relativo de medida del sensor [%]

		Distancia [cm]						
		10	**7,5**	**5**	**0**	**-5**	**-7,5**	**-10**
Ángulo [°]	**5**	26,84%	36,16%	36,68%	0,51%	21,36%	20,00%	20,50%
	10	20,31%	22,05%	11,04%	0,60%	10,11%	6,34%	7,98%
	30	7,51%	6,99%	5,11%	0,39%	3,76%	10,29%	7,24%
	50	4,10%	4,20%	2,29%	0,62%	2,16%	6,99%	4,56%
	70	1,11%	2,06%	1,49%	0,63%	1,25%	1,68%	4,68%
	100	0,80%	0,65%	0,61%	0,66%	0,27%	0,39%	5,49%

Tabla A.4. Error absoluto de medida del sensor [cm]

		Distancia [cm]						
		10	**7,5**	**5**	**0**	**-5**	**-7,5**	**-10**
Ángulo [°]	**5**	1,34	1,81	1,83	0,03	1,07	1,00	1,03
	10	2,03	2,21	1,10	0,06	1,01	0,63	0,80
	30	2,25	2,15	1,53	0,12	1,13	3,09	2,17
	50	2,05	2,22	1,14	0,31	1,08	3,50	2,28
	70	0,78	1,44	1,04	0,44	0,87	1,17	3,27
	100	0,80	0,65	0,61	0,66	0,27	0,39	5,49

Tabla A.5. Medidas *raw* del tracker2D

Posición real objetivo		Distancia obtenida por los sensores	
Distancia [m]	Ángulo [°]	Distancia ultrasonidos 1 - *d1* [cm]	Distancia ultrasonidos 2 - *d2* [cm]
0,1	90	[0,10440 0,1080 0,09980 0,10540 0,100 0,1010 0,09980 0,1080 0,10440 0,10440]	[1,58290 1,57760 1,57880 1,57470 1,58340 1,57420 1,57420 1,57440 1,57860 1,5740]
0,2	90	[0,42840 0,43150 0,42840 0,43160 0,42840 0,43160 0,43270 0,42720 0,43270 0,4270]	[1,58220 1,5820 1,57780 1,58220 1,57810 1,58590 1,57740 1,57830 1,5870 1,57830]
0,5	90	[0,5020 0,50180 0,50640 0,50640 0,50340 0,50220 0,50220 0,50640 0,50240 0,50640]	[0,4990 0,49880 0,50320 0,50320 0,49910 0,50320 0,49880 0,50320 0,4990 0,49910]
0,7	90	[0,68660 0,68680 0,69110 0,68660 0,68660 0,68660 0,68660 0,68650 0,68650 0,68650]	[0,69460 0,69020 0,69020 0,69040 0,69020 0,69040 0,68650 0,68650 0,69020 0,69870]
1	90	[0,97820 0,97820 0,97820 0,97820 0,97870 0,97940 0,9790 0,98260 0,98670 0,97820]	[0,97440 0,97870 0,98280 0,97440 0,97440 0,97870 0,97550 0,97550 0,98280 0,97430]
1,2	90	[1,15460 1,15460 1,15970 1,16310 1,15480 1,16310 1,15570 1,1630 1,15890 1,15460]	[1,14630 1,15960 1,14730 1,14750 1,15510 1,14650 1,15180 1,15190 1,14630 1,14650]
0,1	80	[1,57840 1,58320 1,58220 1,58340 1,59190 1,58730 1,59480 1,5830 1,58290 1,58290]	[1,5820 1,61860 1,58170 1,58170 1,58630 1,58080 1,58150 1,58170 1,58640 1,58150]
0,2	80	[1,57910 1,57910 1,57930 1,58360 1,58680 1,58340 1,57930 1,58360 1,5920 1,57910]	[1,5740 1,58290 1,5740 1,57840 1,57390 1,5830 1,57420 1,57420 1,5730 1,5740]
0,5	80	[0,53528 0,53530 0,53550 0,53550 0,53110 0,53130 0,53010 0,53110 0,53550 0,53090]	[0,49560 0,49540 0,49570 0,49560 0,49570 0,49980 0,49570 0,500 0,49570 0,49570]
0,7	80	[0,68120 0,68990 0,68590 0,69020 0,69430 0,68990 0,68610 0,68580 0,68990 0,68990]	[0,6820 0,67760 0,67880 0,67780 0,67780 0,67780 0,67760 0,67760 0,68170 0,67880]
1	80	[0,98290 0,98290 0,98740 0,98310 0,9840 0,98410 0,99130 0,98680 0,98290 0,98290]	[0,9790 0,9750 0,97920 0,98330 0,9790 0,97610 0,97610 0,98330 0,97510 0,97510]
1,2	80	[1,18680 1,17930 1,18350 1,18780 1,18780 1,18760 1,1880 1,18350 1,19730 1,18760]	[1,15860 1,1540 1,15860 1,15860 1,15860 1,15860 1,15450 1,15960 1,15870 1,15550 1,15430]
0,1	10	[0,10640 0,10760 0,11070 0,11150 0,10660 0,11510 0,11150 0,11050 0,11070 0,11050]	[1,57350 1,57340 1,57340 1,57340 1,58240 1,58220 1,57340 1,57780 1,57790 1,56930]
0,2	10	[0,20940 0,20930 0,20880 0,20820 0,20940 0,20930 0,20930 0,20930 0,20930 0,21250]	[1,56940 1,56940 1,56960 1,57440 1,56960 1,58290 1,56940 1,56840 1,57840 1,57420]
0,5	10	[0,49720 0,50150 0,49690 0,49690 0,49830 0,50150 0,49720 0,49710 0,49690 0,49710]	[0,52260 0,51030 0,51030 0,51030 0,52280 0,51030 0,52290 0,52280 0,51030 0,51020]
0,7	10	[0,70305 0,70310 0,70310 0,69890 0,69890 0,69460 0,70310 0,69890 0,69480 0,69850]	[0,68370 0,68360 0,68760 0,68390 0,68360 0,68360 0,67980 0,68390 0,68360 0,68360]
1	10	[0,98310 0,97480 0,98740 0,97990 0,9750 0,97460 0,9790 0,9790 0,97990 0,98410]	[0,99940 0,9870 0,9870 0,99640 0,9880 0,99540 0,99540 0,9870 0,98680 1,060]
1,2	10	[1,1870 1,1880 0 1,18050 1,2070 1,19660 1,18810 1,19650 1,18810 1,19240 1,1880]	[1,17180 1,17180 1,16860 1,16740 1,17270 1,16740 1,16740 1,16760 1,17610 1,16740]

Tabla A.6. Medidas error del tracker2D

Medida real		Resultado medidas tracker2D										Medida	Error absoluto
Azimut [°]	90	79,44	92,35	81,06	99,71	82,35	91,06	79,31	82,35	80,39	89,58	85,76	4,24
Distancia [cm]	5	4,42	4,42	4,48	4,43	4,42	4,48	4,42	4,42	4,45	4,43	4,44	0,56
Azimut [°]	90	80,78	84,95	90,55	90,55	90,46	90,71	90,71	85,98	75,98	81,83	86,25	3,75
Distancia [cm]	50	49,42	50,28	50,29	50,29	50,34	50,26	50,29	50,29	50,70	50,29	50,24	0,24
Azimut [°]	90	90,00	89,77	89,77	99,33	94,25	80,09	79,86	80,09	83,68	84,95	87,18	2,82
Distancia [cm]	100	98,08	98,07	98,07	98,48	99,23	98,51	98,50	98,51	98,36	98,42	98,42	1,58
Azimut [°]	95	93,44	95,41	88,88	100,16	87,63	84,53	97,55	84,53	92,09	84,53	90,88	4,12
Distancia [cm]	5	6,51	5,54	5,82	6,23	5,77	6,01	6,22	6,01	6,45	6,01	6,06	1,06
Azimut [°]	95	93,55	98,87	98,87	98,87	101,78	95,78	93,55	95,59	98,93	96,37	97,22	2,22
Distancia [cm]	50	51,04	51,26	51,26	51,26	51,05	51,05	51,04	51,05	51,25	51,67	51,19	1,19
Azimut [°]	95	99,86	95,28	95,75	93,90	90,23	92,18	93,67	88,52	94,72	95,51	93,96	1,04
Distancia [cm]	100	98,28	98,44	98,87	98,50	98,32	98,44	98,24	98,03	98,07	98,55	98,37	1,63
Azimut [°]	85	78,75	70,05	83,05	68,75	87,20	68,75	89,40	90,05	89,06	67,20	79,23	5,77
Distancia [cm]	5	4,52	4,74	4,45	4,82	4,97	4,52	4,98	5,45	4,53	4,47	4,74	0,26
Azimut [°]	85	86,02	80,20	81,59	85,03	85,03	86,10	84,48	80,20	86,10	85,05	83,98	1,02
Distancia [cm]	50	50,73	51,17	50,72	50,95	50,96	51,39	51,22	51,15	50,94	50,90	51,02	1,02
Azimut [°]	85	84,92	80,01	84,81	84,37	88,83	78,31	84,58	83,45	86,28	89,07	84,46	0,54
Distancia [cm]	100	97,70	97,18	97,92	98,96	98,54	98,34	98,02	97,76	98,18	97,69	98,03	1,97

Tabla A.7. Medidas configuraciones del tracker2D

Algoritmo 3							
Posición real objetivo		Medida obtenida					
		Configuración 1		Configuración 2		Configuración 3	
Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]
0,1	90	0,08	85,75	0,08	87,65	0,10	89,32
0,2	90	0,16	84,55	0,18	86,56	0,20	88,75
0,5	90	0,47	83,75	0,47	88,56	0,50	87,83
0,7	90	0,72	87,64	0,71	89,55	0,72	89,72
1	90	0,97	89,64	0,98	91,33	0,98	90,05
1,2	90	1,26	88,12	1,12	89,86	1,09	89,87
0,1	80	0,12	78,13	0,08	78,63	0,11	78,55
0,2	80	0,16	77,13	0,17	77,37	0,19	78,67
0,5	80	0,51	78,63	0,49	78,32	0,48	79,52
0,7	80	0,70	77,65	0,71	79,33	0,69	81,22
1	80	1,06	82,64	1,08	82,16	1,06	80,75
1,2	80	1,17	81,33	1,22	80,16	1,22	80,05
0,1	100	0,11	96,51	0,10	96,65	0,10	98,32
0,2	100	0,17	95,36	0,17	95,97	0,19	98,54
0,5	100	0,47	93,27	0,51	98,99	0,51	99,75
0,7	100	0,69	99,61	0,69	98,63	0,70	100,65
1	100	0,98	102,64	0,99	101,66	1,02	99,25
1,2	100	1,18	101,37	1,19	100,27	1,20	99,88

Tabla A.8. Error relativo configuraciones del tracker2D

Algoritmo 3							
Posición real objetivo		Error relativo					
		Configuración 1		Configuración 2		Configuración 3	
Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]
0,1	90	21,00%	4,72%	21,37%	2,61%	4,00%	0,76%
0,2	90	17,70%	6,06%	8,20%	3,82%	1,00%	1,39%
0,5	90	6,02%	6,94%	6,03%	1,60%	0,80%	2,41%
0,7	90	-2,86%	2,62%	-1,94%	0,50%	-2,14%	0,31%
1	90	3,20%	0,40%	2,37%	-1,47%	1,80%	-0,05%
1,2	90	-5,00%	2,09%	7,00%	0,15%	9,58%	0,14%
0,1	80	-20,00%	2,34%	21,10%	1,71%	-10,00%	1,81%
0,2	80	17,57%	3,58%	16,00%	3,29%	7,00%	1,66%
0,5	80	-2,46%	1,71%	2,06%	2,10%	4,40%	0,60%
0,7	80	0,18%	2,94%	-1,86%	0,84%	1,57%	-1,53%
1	80	-5,80%	-3,30%	-8,00%	-2,70%	-5,60%	-0,94%
1,2	80	2,92%	-1,66%	-1,67%	-0,20%	-1,25%	-0,06%
0,1	100	-13,60%	3,49%	4,00%	3,35%	3,69%	1,68%
0,2	100	15,19%	4,64%	15,50%	4,03%	6,51%	1,46%
0,5	100	6,84%	6,73%	-2,60%	1,01%	-2,52%	0,25%
0,7	100	2,09%	0,39%	0,80%	1,37%	-0,37%	-0,65%
1	100	1,68%	-2,64%	1,20%	-1,66%	-1,60%	0,75%
1,2	100	1,76%	-1,37%	0,83%	-0,27%	0,31%	0,12%

Tabla A.9. Resultados algoritmos del tracker2D

Configuración 3							
Posición real objetivo		Medida obtenida					
		Algoritmo 1		Algoritmo 2		Algoritmo 3	
Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]
0,1	90	0,09	77,27	0,09	77,27	0,10	89,32
0,2	90	0,21	76,27	0,21	74,27	0,20	88,75
0,5	90	0,50	77,85	0,50	78,25	0,50	87,83
0,7	90	0,68	81,62	0,68	82,75	0,72	88,22
1	90	0,97	82,83	0,97	87,83	0,98	87,72
1,2	90	1,15	87,25	1,17	85,25	1,19	85,87
0,1	80	0,12	72,25	0,12	75,01	0,11	78,55
0,2	80	0,18	73,71	0,18	75,12	0,19	78,67
0,5	80	0,47	74,53	0,48	75,96	0,48	79,52
0,7	80	0,68	76,25	0,67	77,66	0,69	81,22
1	80	0,97	74,79	0,96	76,20	1,06	80,75
1,2	80	1,17	77,26	1,19	76,20	1,12	80,75
0,1	100	0,12	95,37	0,99	95,95	0,10	98,32
0,2	100	0,17	95,64	0,20	96,20	0,19	98,54
0,5	100	0,46	96,87	0,53	97,42	0,51	99,75
0,7	100	0,67	97,74	0,70	98,31	0,71	100,65
1	100	0,95	96,35	0,99	96,92	1,02	99,25
1,2	100	1,16	97,22	1,21	97,85	1,18	99,88

Tabla A.10. Error relativo algoritmos del tracker2D

Configuración 3							
Posición real objetivo		Medida obtenida					
		Algoritmo 1		Algoritmo 2		Algoritmo 3	
Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]	Distancia [m]	Ángulo [°]
0,1	90	13,00%	14,14%	13,00%	14,14%	4,00%	0,76%
0,2	90	-5,50%	15,26%	-5,50%	17,48%	1,00%	1,39%
0,5	90	0,80%	13,50%	0,80%	13,06%	0,80%	2,41%
0,7	90	3,14%	9,31%	3,14%	8,06%	-2,14%	1,98%
1	90	3,30%	7,97%	3,30%	2,41%	1,80%	2,53%
1,2	90	4,00%	3,06%	2,75%	5,28%	1,25%	4,59%
0,1	80	-21,00%	9,69%	-21,00%	6,23%	-10,00%	1,81%
0,2	80	8,00%	7,86%	8,00%	6,09%	7,00%	1,66%
0,5	80	6,36%	6,84%	4,62%	5,05%	4,40%	0,60%
0,7	80	2,21%	4,69%	3,69%	2,92%	1,57%	-1,53%
1	80	3,30%	6,52%	3,69%	4,75%	-5,60%	-0,94%
1,2	80	2,75%	3,42%	1,25%	4,75%	7,08%	-0,94%
0,1	100	-15,25%	4,63%	-8,90%	4,05%	3,69%	1,68%
0,2	100	14,10%	4,36%	0,72%	3,80%	6,51%	1,46%
0,5	100	8,79%	3,13%	-5,60%	2,58%	-2,52%	0,25%
0,7	100	3,96%	2,26%	0,19%	1,69%	-1,80%	-0,65%
1	100	4,52%	3,65%	1,46%	3,08%	-1,60%	0,75%
1,2	100	3,22%	2,78%	-1,08%	2,15%	1,33%	0,12%

Tabla A.11. Error cometido en plano horizontal en el tracker3D

Posición real del objetivo		Posición calculada		Error absoluto	
Distancia [cm]	Azimut [°]	Distancia [cm]	Azimut [°]	Distancia [cm]	Azimut [°]
50	-5	49,86	-5,235	0,14	0,235
80	0	79,58	0,885	0,42	-0,885
120	0	120,37	-0,2795	-0,37	0,2795
100	5	99,67	5,135	0,33	-0,135

Tabla A.12. Resultado obtenido en plano vertical

Posición real del objetivo		Posición calculada		Error absoluto	
Distancia [cm]	Azimut [°]	Distancia [cm]	Azimut [°]	Distancia [cm]	Azimut [°]
50	-5	51,21	-5,58	-1,21	0,58
50	0	49,86	-1,27	0,14	1,27
50	5	48,12	6,21	1,88	-1,21

APÉNDICE B. Código Matlab

B.1. Rutinas para probar los sensores

Para poder estudiar el comportamiento de los sensores y del sistema de seguimiento, son necesarios diferentes programas en Matlab para obtener y procesar las medidas.

B.1.1. Rutina para probar un sensor

El programa a definir en Matlab para obtener la distancia obtenida por el sensor, desde Arduino por puerto serie, es el siguiente:

```
%% DEFINIMOS PARÁMETROS ARDUINO
%DEFINIMOS Y ABRIMOS SERIAL DE LECTURA
s = serial('COM6'); %especificamos puerto de lectura
s.BaudRate = 9600;  %ratio  [Hz]
fopen(s);

i=1;
muestras=1000;
while(i<=muestras)
    datos = fscanf(s); %lectura de datos del puerto
    [radio] = strtok(datos); %distancia

    radio = str2num(radio); %obtenemos el radio

    dist(i) = radio;

    i = i +1;
    pause(1);
    fprintf('Progreso : %f %% dist : %f\n',i/muestras*100,radio);
end

distancia = 100;
angulo = 5;

figure;
plot(dist);
title(['Distancia [cm] - ',num2str(distancia),' cm / ',
num2str(angulo),'°'])

fclose(s);
```

B.2. Estudio del tracker2D

B.2.1. Código de Arduino

Para obtener las lecturas de los sensores y enviarlas por puerto serie:

```
#ifdef ESTUDIO_2D

double dist1 = distSENSOR1(); delay(20); //Medidas de los sensores
double dist2 = distSENSOR2(); delay(20); //Elimina ecos (20 ms = 6,8 m)
//Enviamos los datos por puerto serie
Serial.print(" ");
Serial.print(dist1,2); Serial.print(" ");
Serial.print(dist2,2); Serial.println(" ");

#endif
```

B.2.2. Código de Matlab

El estudio de las configuraciones posibles de los sensores y de los diferentes algoritmos matemáticos para posicionamiento, en el caso del sistema de seguimiento bidimensional, se ha realizado con el siguiente programa:

```
%% PRUEBA DE ALGORITMOS
%Configuración 1, 2 y 3.
%% INPUTS
a = input('Introduzca la configuración del tracker: 1, 2 o 3\n');

if (a == 1)
%Configuración 1 (Ultrasonidos colocados normal: RX_a *centro* TX_a
Rx_b Tx_b)
load('C:\Users\Artur\Google
Drive\TFG\Código\raw_tracker_data\tracker_conf1.mat');
elseif (a == 2)
%Configuración 2 (Ultrasonidos 2 está invertido: RX_a *centro* TX_a
Tx_b Rx_b)
load('C:\Users\Artur\Google
Drive\TFG\Código\raw_tracker_data\tracker_conf2.mat');
else
%Configuración 3 (Ultrasonidos colocados normal: RX_a TX_a *centro*
Rx_b Tx_b)
load('C:\Users\Artur\Google
Drive\TFG\Código\raw_tracker_data\tracker_centered.mat');
end

b = input('Introduzca la distancia (0.1, 0.2, 0.5, 0.7, 1, 1.2)
[m]:\n');
c = input('Introduzca el angulo azimut (80, 90, 100) [°]:\n');

d= b; a=c;
for i=1:length(tracker)
    %Buscamos distancia
```

```
    if (tracker(i).dist==d)
    j(i) = 2;
    else
    j(i) = 0;
    end
     %Buscamos angulo
    if (tracker(i).angle==a)
    k(i) = 1;
    else
    k(i) = 0;
    end
end

pos = j-k;
pos = find(pos==1);

dist1 = mean(tracker(pos).d1);
dist2 = mean(tracker(pos).d2);

%COMPARAMOS ALGORITMOS
[azimut1, distancia1] = polarPOSITION(dist1, dist2); %Circulos 1
[azimut2, distancia2] = polarPOSITIONv2(dist1, dist2); %Circulos 2
[azimut3, distancia3] = polarPOSITIONv3(dist1, dist2); %Elipses

%Calculamos valores reales (ideales)
d = 5/100;
x = b * cos(c*pi/180);
y = b * sin(c*pi/180);
dreal = sqrt(x^2+y^2);
areal = atan2d(y,x);

fprintf('DISTANCIA  : %f m ***** ANGULO  : %f °\n',dreal,areal);
fprintf('DISTANCIA 1: %f m ***** ANGULO 1: %f
°\n',distancia1,azimut1);
fprintf('DISTANCIA 2: %f m ***** ANGULO 2: %f
°\n',distancia2,azimut2);
fprintf('DISTANCIA 3: %f m ***** ANGULO 3: %f °\n',distancia3,90-
azimut3);
```

En el programa anterior, recorremos la matriz de datos en bruto, escogiendo la configuración de sensores a utilizar con el objetivo en un cierto punto y el algoritmo matemático a aplicar para obtener la distancia y el ángulo al mismo.

B.3. Estudio del tracker3D

B.3.1. Código de Arduino

Para obtener las lecturas de los sensores y enviarlas por puerto serie:

```
  #ifdef ESTUDIO_3D

  double dist1 = distSENSOR1(); delay(20); //Medidas de los sensores
  double dist2 = distSENSOR2(); delay(20); //Elimina ecos (20 ms = 6,8 m)
  double dist3 = distSENSOR3(); delay(20);
  double dist4 = distSENSOR4(); delay(20);
  //Enviamos los datos por puerto serie
```

```
  Serial.print(" ");  Serial.print(dist1,2);Serial.print(" ");
  Serial.print(dist2,2); Serial.print(" ");
  Serial.print(dist3,2); Serial.print(" "); Serial.println(dist4,2);

  #endif
```

B.3.2. Código de Matlab

Obtenemos los datos enviados por el puerto serie.

```
%% DEFINIMOS PARÁMETROS ARDUINO
%DEFINIMOS Y ABRIMOS SERIAL DE LECTURA
s = serial('COM3'); %especificamos puerto de lectura
s.BaudRate = 9600;  %ratio  [Hz]
fopen(s);

i=1;
muestras=1000;
while(i<=muestras)
    datos = fscanf(s); %lectura de datos del puerto
    [d1,remain] = strtok(datos);
    [d2,remain] = strtok(remain);
    [d3,remain] = strtok(remain);
    [d4,remain] = strtok(remain);

    d1 = str2num(d1); d2 = str2num(d2);
    d3 = str2num(d3); d4 = str2num(d4);

    dist1(i) = d1; dist2(i) = d2;
    dist3(i) = d3; dist4(i) = d4;
    i = i + 1;

    pause(0.15);
end

%Datos reales
distancia = 50;
angulo = 10;
%Datos obtenidos
j=3; %Guardamos datos en una estructura
tracker3D(j).dist=distancia;
tracker3D(j).angle=angulo;
tracker3D(j).d1=dist1;
tracker3D(j).d2=dist2;
tracker3D(j).d3=dist4;
tracker3D(j).d4=dist3;

fclose(s);


%% RESULTADOS Y GRÁFICOS

medidasH = tracker3D; %Escogemos las medidas horizontales

%***HORIZONTAL***
%PUNTO 100 cm - 0 deg
```

```
dist1 = mean(medidasH(1).d1/100); dist2 = mean(medidasH(1).d2/100);
[a1,d1]=polarPOSITIONv3(dist1, dist2);

%PUNTO 50 cm - 0 deg
dist1 = mean(medidasH(2).d1/100); dist2 = mean(medidasH(2).d2/100);
[a2,d2]=polarPOSITIONv3(dist1, dist2);


%% HISTOGRAMAS

%% 50 cm y 5°
dist1 = medidasH(1).d1;

for i=1:length(dist1)
   dist1 = medidasH(1).d1(i)/100;
   dist2 = medidasH(1).d2(i)/100;

   [a1(i),d1(i)]=polarPOSITIONv3(dist1, dist2);
end

figure;
histogram(a1,10); xlabel("Ángulo [°]"); ylabel("Número de muestras");

figure;
histogram(d1,10); xlabel("Distancia [cm]"); ylabel("Número de
muestras");
```

APÉNDICE C. Código sistema bidimensional

C.1. Radar sonar bidimensional

Una de las aplicaciones interesantes del sonar bidimensional es un detector de objetos en 360º, típicamente usado en embarcaciones y submarinos. En el entorno de Arduino debemos crear un *sketch* para mover el motor en pasos incrementales de 1º y obtener con el sensor de ultrasonidos la distancia leída en cada uno de los pasos. Para ello escribimos el siguiente programa:

```
const float vprop = 34000; //cm/s
float distancia = 0.00;
int pos = 1;

// Configuramos los pines del sensor Trigger y Echo
const int PinTrig = 8, PinEcho = 9;

void setup()
{
  // Iniciamos el monitor serie para mostrar el resultado
  Serial.begin(9600);
  // Ponemos el pin Trig en modo salida
  pinMode(PinTrig, OUTPUT);
  // Ponemos el pin Echo en modo entrada
  pinMode(PinEcho, INPUT);
 }

void loop()
{
  distancia = dameDISTANCIA(PinTrig, PinEcho); //Lectura del sensor

  //OUTPUT del serial: posicion (theta) y distancia (radio)
  Serial.print(pos); Serial.print(" "); Serial.println(distancia);

  moveMOTOR(1); //Movemos el motor 1º
  pos = pos + 1;

  if (pos > 360) //Vuelve a empezar
  pos = 1;

  delay(300);
}
```

Donde las funciones `dameDISTANCIA()` y `moveMOTOR()` han sido definidas en el apartado 2.3.1. Entonces en MATLAB podemos definir un gráfico para mostrar un radar 360º y recibir por el puerto serie el ángulo del motor y la distancia leída por el sensor de ultrasonidos en ese punto.

Primero definimos el gráfico:

```
figure();
whitebg('black'); %Color de fondo del radar

rmax = 150;
%CIRCULOS
R = 10:10:rmax;  %Radios cada 10 cm hasta rmax cm
theta = linspace(0, 2*pi, 500); %Generamos 500 puntos entre 0° y 360°

for i=1:length(R)
    x = R(i)*cos(theta);
    y = R(i)*sin(theta);
    plot(x,y,'Color', [0 1 0] ,'LineWidth',1);
    hold on;
end

%RADIALES
%Ejes radiales
for i=1:360
    [x, y] = pol2cart(i*pi/180,rmax);
    h(i) = plot([0,x],[0,y],'g','LineWidth',1);
    hold on;
end

%Ejes radiales anchos (cada 45 grados)
for i=0:8
    [x, y] = pol2cart(i*45*pi/180,rmax);
    plot([0,x],[0,y],'y','LineWidth',1);
    hold on;
end
```

Y obtenemos los datos del sensor y del motor (desde Arduino) por puerto serie:

```
%DEFINIMOS Y ABRIMOS SERIAL DE LECTURA
s = serial('COM6'); %Especificamos puerto serie de lectura
s.BaudRate = 9600;  %Frecuencia del puerto  [Hz]
fopen(s);

for i=1 %Loop infinito

    datos = fscanf(s); %lectura de datos del puerto
    [theta, radio] = strtok(datos); %extremos angulo y distancia

    theta = str2num(theta); %obtenemos el ángulo del motor
    radio = str2num(radio); %obtenemos la distancia obtenida

    if (radio > rmax) %FUERA de rango
        radio = 0;
    end

    delete(h(theta)); %Eliminamos radiales verdes (efecto radar)

    [x0, y0] = pol2cart(theta*pi/180, 0); %coordenadas origen
    [x1, y1] = pol2cart(theta*pi/180, radio); %coordenadas objetivo

    t(i) = theta; %guardamos ángulo
    r(i) = radio; %guardamos radio
```

```
    plot([x0,x1],[y0,y1],'r','LineWidth',2); %Radial rojo
    hold on;

    drawnow;
    i = i +1;
end
```

La rutina anterior dibuja un radial rojo de longitud igual a la distancia obtenida por el sensor de ultrasonidos considerando el ángulo de posición del motor.

APÉNDICE D. Código sistema tridimensional

D.1. Sistema de control completo

Recordando la estructura del código en el apartado 2.3.3.2, empezaremos definiendo las clases utilizadas en el sistema de control.

D.1.1. Clase *Tracker*

Partimos del archivo cabecera (Tracker.h) declarando todas las variables y funciones correspondientes:

```
#ifndef TRACKER_H
#define TRACKER_H

class Tracker
{
public:

    void setupSensor(); //Inicia un sensor
    void setupTracker2D(); //Inicia dos sensores: tracker2D
    void setupTracker3D(); //Inicia cuatro sensores: tracker3D
    void setupLED(int status); //Enciende o apaga el LED láser

    void setupMotor1();  //Inicia un motor: tracker2D
    void setupMotor2();  //Inicia dos motor: tracker3D

    //SENSOR ULTRASONIDO (una función por sensor: 1, 2, 3 y 4)
    double getTIME1(); //Devuelve el tiempo de vuelo
    double getTIME2();
    double getTIME3();
    double getTIME4();

    double getDISTANCE1(); //Calcula la distancia al objetivo
    double getDISTANCE2();
    double getDISTANCE3();
    double getDISTANCE4();

    //MOTOR
    void moveMOTOR(double grados, int motorID); //Mueve el motor

    //DECLARACIÓN DE PINES DE LOS SENSORES
    int GND1 = 14;
    int ECHO1 = 15;
    int TRIG1 = 16;
    int VCC1 = 17;
```

```
    int GND2 = 18;
    int ECHO2 = 19;
    int TRIG2 = 20;
    int VCC2 = 21;

    int GND3 = 7;
    int ECHO3 = 6;
    int TRIG3 = 5;
    int VCC3 = 4;

    int GND4 = 8;
    int ECHO4 = 9;
    int TRIG4 = 10;
    int VCC4 = 11;

    //PINES DEL MOTOR1 - MOTOR HORIZONTAL (azimut)
    int M1_IN1 = 30;
    int M1_IN2 = 32;
    int M1_IN3 = 34;
    int M1_IN4 = 36;

    //PINES DEL MOTOR2 - MOTOR VERTICAL (elevación)
    int M2_IN1 = 22;
    int M2_IN2 = 24;
    int M2_IN3 = 26;
    int M2_IN4 = 28;

    //PINES DEL LED LÁSER
    int VCCLED = 3;
    int GNDLED = 2;

};

#endif
```

Seguidamente definimos las funciones y procedimientos de la clase *Tracker* en el archivo (Tracker.cpp):

```
#include "Tracker.h"
#include "Arduino.h"

void Tracker::setupSensor()
{
  pinMode(this->TRIG1, OUTPUT);
  pinMode(this->ECHO1, INPUT);

}
```

```
void Tracker::setupTracker2D()
{
  pinMode(this->TRIG1, OUTPUT);
  pinMode(this->TRIG2, OUTPUT);
  pinMode(this->ECHO1, INPUT);
  pinMode(this->ECHO2, INPUT);

  pinMode(this->VCC1, OUTPUT);
  pinMode(this->VCC2, OUTPUT);
  pinMode(this->GND1, OUTPUT);
  pinMode(this->GND2, OUTPUT);

  digitalWrite(this->VCC1, HIGH);
  digitalWrite(this->VCC2, HIGH);
  digitalWrite(this->GND1, LOW);
  digitalWrite(this->GND2, LOW);

}

void Tracker::setupTracker3D()
{
  pinMode(this->TRIG1, OUTPUT);
  pinMode(this->TRIG2, OUTPUT);
  pinMode(this->TRIG3, OUTPUT);
  pinMode(this->TRIG4, OUTPUT);

  pinMode(this->ECHO1, INPUT);
  pinMode(this->ECHO2, INPUT);
  pinMode(this->ECHO3, INPUT);
  pinMode(this->ECHO4, INPUT);

  pinMode(this->VCC1, OUTPUT);
  pinMode(this->VCC2, OUTPUT);
  pinMode(this->VCC3, OUTPUT);
  pinMode(this->VCC4, OUTPUT);
  pinMode(this->GND1, OUTPUT);
  pinMode(this->GND2, OUTPUT);
  pinMode(this->GND3, OUTPUT);
  pinMode(this->GND4, OUTPUT);

  digitalWrite(this->VCC1, HIGH);
  digitalWrite(this->VCC2, HIGH);
  digitalWrite(this->VCC3, HIGH);
  digitalWrite(this->VCC4, HIGH);
  digitalWrite(this->GND1, LOW);
  digitalWrite(this->GND2, LOW);
  digitalWrite(this->GND3, LOW);
  digitalWrite(this->GND4, LOW);
}
```

```
void Tracker::setupMotor1()
{
  pinMode(this->M1_IN1, OUTPUT);
  pinMode(this->M1_IN2, OUTPUT);
  pinMode(this->M1_IN3, OUTPUT);
  pinMode(this->M1_IN4, OUTPUT);

}

void Tracker::setupMotor2()
{
  pinMode(this->M2_IN1, OUTPUT);
  pinMode(this->M2_IN2, OUTPUT);
  pinMode(this->M2_IN3, OUTPUT);
  pinMode(this->M2_IN4, OUTPUT);

}

void Tracker::setupLED(int onoff)
{
  pinMode(this->VCCLED, OUTPUT);
  pinMode(this->GNDLED, OUTPUT);

  if (onoff == 1){
  digitalWrite(this->VCCLED, HIGH);
  digitalWrite(this->GNDLED, LOW);
  }
  else {
  digitalWrite(this->VCCLED, LOW);
  digitalWrite(this->GNDLED, LOW);
  }

}

//*****ULTRASONIC SENSOR*******

//****TIME*****
double getTIME(unsigned int TRIG, unsigned int ECHO){

  //Desactivamos TRIGGER
  digitalWrite(TRIG, LOW);
  delayMicroseconds(10);

  //Activamos TRIGGER -> EMITIMOS PULSO
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);
```

```
//Tiempo que ECHO esta en estado HIGH
  double tiempo = pulseIn(ECHO, HIGH);
  return tiempo;
}

double Tracker::getTIME1(){
  return getTIME(this->TRIG1, this->ECHO1);
}
double Tracker::getTIME2(){
  return getTIME(this->TRIG2, this->ECHO2);
}
double Tracker::getTIME3(){
  return getTIME(this->TRIG3, this->ECHO3);
}
double Tracker::getTIME4(){
  return getTIME(this->TRIG4, this->ECHO4);
}

//****DISTANCIA*****
const float vprop = sqrt(1.4*8.314*293.15/0.029)*100; //cm/s

double getDISTANCE(unsigned int TRIG, unsigned int ECHO){

  double tiempo = getTIME(TRIG, ECHO);

  return  tiempo* vprop / 1000000 / 2;
}

double Tracker::getDISTANCE1(){
  return getDISTANCE(this->TRIG1, this->ECHO1);
}
double Tracker::getDISTANCE2(){
  return getDISTANCE(this->TRIG2, this->ECHO2);
}
double Tracker::getDISTANCE3(){
  return getDISTANCE(this->TRIG3, this->ECHO3);
}
double Tracker::getDISTANCE4(){
  return getDISTANCE(this->TRIG4, this->ECHO4);
}

//****STEPPER MOTOR*******

//Secuencia de excitacion de las bobinas
int sec [8][4] =
{
  {1, 0, 0, 0},
  {1, 1, 0, 0},
  {0, 1, 0, 0},
```

```
  {0, 1, 1, 0},
  {0, 0, 1, 0},
  {0, 0, 1, 1},
  {0, 0, 0, 1},
  {1, 0, 0, 1}
};

float ratio = 8*51.2/36;
float tot_pasos;
void Tracker::moveMOTOR(double grados, int motorID)
{
  tot_pasos = abs(grados)*ratio;
  int pasos, i = 0;

for (int pasos = 0; pasos < tot_pasos; pasos++)
    {
      if (grados>=0){
        if (motorID == 1){
          digitalWrite(this->M1_IN1, sec[i][0]);
          digitalWrite(this->M1_IN2, sec[i][1]);
          digitalWrite(this->M1_IN3, sec[i][2]);
          digitalWrite(this->M1_IN4, sec[i][3]);
        }
        else{
          digitalWrite(this->M2_IN1, sec[i][0]);
          digitalWrite(this->M2_IN2, sec[i][1]);
          digitalWrite(this->M2_IN3, sec[i][2]);
          digitalWrite(this->M2_IN4, sec[i][3]);
        }
      }
      else{
        if (motorID == 1){
          digitalWrite(this->M1_IN1, sec[i][3]);
          digitalWrite(this->M1_IN2, sec[i][2]);
          digitalWrite(this->M1_IN3, sec[i][1]);
          digitalWrite(this->M1_IN4, sec[i][0]);
        }
        else{
          digitalWrite(this->M2_IN1, sec[i][3]);
          digitalWrite(this->M2_IN2, sec[i][2]);
          digitalWrite(this->M2_IN3, sec[i][1]);
          digitalWrite(this->M2_IN4, sec[i][0]);
        }
      }
        i++;
        if (i==8)
        i = 0;

      delay(2);
```

```
    }
}
```

D.1.2. Clase *Maths*

Por otro lado, tenemos la clase *Maths* definida en el archivo Maths.h:

```
#ifndef MATHS_H
#define MATHS_H

class Maths
{
public:
    //MATHS
    //****
    double computeAngleV1(double d1, double d2); //TRIGONOMETRIA
    double computeAngleV2(double d1, double d2); //CIRCULOS
    double computeAngleV3(double d1, double d2); //ELIPSES

    double dist = 7.0/100; //Distancia entre sensores [m]
};

#endif
```

En el archivo Maths.cpp se definen las funciones para aplicar los tres algoritmos.

```
#include "Maths.h"
#include <math.h>

//polarPositionV1
double Maths::computeAngleV1(double d1, double d2){
    double d = this->dist;

    //RELACIONES TRIGONOMÉTRICAS
    //Problema planteado considerando posiciones tal que x > 0 (derecha)
    double h = sqrt(pow(d,2) - pow((pow(d,2) + pow(d1,2) - pow(d2,2))/pow
(2*d1,2),2));
    double a_rad = asin(h/d);

    //Diferenciamos derecha/izquierda
    if (d2>d1){a_rad = M_PI - a_rad ;}

    //POSICION (cartesiana) respecto al punto medio
    double x = d1 * cos(a_rad) - d/2;
    double y = d1 * sin(a_rad);

    //OBTENEMOS ANGULO Y DISTANCIA respecto al punto medio
    double azimut_rad = atan2(y,x);
```

```
    double azimut_deg = azimut_rad * 180 / M_PI ;
    //double distancia_m = sqrt(pow(x,2) + pow(y,2));

    double rel_azimut_deg = 90 - azimut_deg;
    return rel_azimut_deg;
}

//polarPositionV2
double Maths::computeAngleV2(double d1, double d2){
    double d = this->dist;

    //POSICION (cartesiana) respecto al punto medio
    double x = (pow(d1,2)-pow(d2,2))/(2*d);
    double y = sqrt(pow(d1,2) - pow( x + d/2,2));

    double azimut_rad = atan2(y,x); //tiene en cuenta los signos

    double azimut_deg = azimut_rad * 180 / M_PI;
    //double distancia_m = sqrt(pow(x,2) + pow(y,2));

    double rel_azimut_deg = 90 - azimut_deg;
    return rel_azimut_deg;
}

//polarPositionV3
double Maths::computeAngleV3(double d1, double d2){

  //PARAMETERS
  double a = this->dist/2;
  double b = a;
  double c = this->dist/2;
  double d = a/2 + c + b/2;

    //Computing x coordinate
    double mxp1=(pow(d1,2)-pow(d2,2))/2/d;

    //Computing y coordinate
    double myp1=sqrt(pow(d1,2)-pow(mxp1+d*0.5,2));

    //Position vector
    double mA1=-(sqrt(pow(mxp1+0.5*d+0.5*a,2)+pow(myp1,2))+
sqrt(pow(mxp1+0.5*d-0.5*a,2)+pow(myp1,2))-2*d1);
    double mA2=-(sqrt(pow(mxp1-0.5*d+0.5*b,2)+pow(myp1,2))+sqrt(pow(mxp1-
0.5*d-0.5*b,2)+pow(myp1,2))-2*d2);

    //Matrix
    double mB1=(mxp1+0.5*(d+a))/sqrt(pow(mxp1+0.5*d+0.5*a,2)+pow(myp1,2))
+(mxp1+0.5*(d-a))/sqrt(pow(mxp1+0.5*d-0.5*a,2)+pow(myp1,2));
```

```
    double mB2=(mxp1-0.5*(d-b))/sqrt(pow(mxp1-
0.5*d+0.5*b,2)+pow(myp1,2))+(mxp1-0.5*(d+b))/sqrt(pow(mxp1-0.5*d-
0.5*b,2)+pow(myp1,2));
    double mC1=myp1/sqrt(pow(mxp1+0.5*d+0.5*a,2)+pow(myp1,2))+myp1/sqrt(p
ow(mxp1+0.5*d-0.5*a,2)+pow(myp1,2));
    double mC2=myp1/sqrt(pow(mxp1-
0.5*d+0.5*b,2)+pow(myp1,2))+myp1/sqrt(pow(mxp1-0.5*d-
0.5*b,2)+pow(myp1,2));

    //Finding determinant
    double mdet=mB1*mC2-mC1*mB2;

    //Computing x coordinate
    double mxp2=mxp1+(mC2*mA1-mC1*mA2)/mdet;

    //Computing y coordinate
    double myp2=myp1+(-mB2*mA1+mB1*mA2)/mdet;

    //Computing distance to solution
    double mrp2=sqrt(pow(myp2,2)+pow(myp2,2));

    //Computing angle to solution
    double map2=atan2(mxp2,myp2);

    double rel_azimut_deg = map2* 180 / M_PI;
    //double distancia_m = mrp2;

    return rel_azimut_deg;
}
```

D.1.3. Archivos de configuración

Para enlazar las clases con el *sketch* principal de Arduino necesitamos dos archivos para declarar la clase/objeto y "traducir" las funciones entre ambos entornos.

Por un lado, tenemos el archivo Tracker_conf.h:

```
#include "Tracker.h"

Tracker tracker;

void setupSensor(){
    tracker.setupSensor();
}

void setupTRACKER2D(){
    tracker.setupTracker2D();
}
```

```
void setupTRACKER3D(){
    tracker.setupTracker3D();
}

void setupMOTOR1(){
    tracker.setupMotor1();
}

void setupMOTOR2(){
    tracker.setupMotor2();
}

void setupLED(int onoff){
    tracker.setupLED(onoff);
}

double distSENSOR1(){
    return tracker.getDISTANCE1();
}

double distSENSOR2(){
    return tracker.getDISTANCE2();
}

double distSENSOR3(){
    return tracker.getDISTANCE3();
}

double distSENSOR4(){
    return tracker.getDISTANCE4();
}

void moverMOTOR1(double grados){
    tracker.moveMOTOR(grados,1);
}

void moverMOTOR2(double grados){
    tracker.moveMOTOR(grados,2);
}
```

Y por otro lado, tenemos el archivo Maths_conf.h:

```
#include "Maths.h"

Maths maths;

double computeAngleV1(double d1, double d2){
    return maths.computeAngleV1(d1, d2);
```

```
}

double computeAngleV2(double d1, double d2){
    return maths.computeAngleV2(d1, d2);
}

double computeAngleV3(double d1, double d2){
    return maths.computeAngleV3(d1, d2);
}
```

D.1.4. Programa principal de Arduino

El programa o *sketch* principal que será ejecutado por la placa Arduino será:

```
#include "Tracker_conf.h"
#include "Maths_conf.h"
#include "Arduino.h"

//#define MATHS //Para testear algoritmos de cálculo
//#define MOTOR //Para testear stepper Motor 28BYJ

//#define TRACKER2D //Dos ultrasonidos horizontales (Ultrasonidos 1 y 2)
//#define MOTOR2D

#define TRACKER3D //Cuatro ultrasonidos (Ultrasonidos 1, 2, 3 y 4)
#define MOTOR3D

#define LED

//Posiciones angular máximas y mínimas posibles (LIMITACIÓN FÍSICA)
double actualH=0; double minH=-45; double maxH=45;
double actualV=0; double minV=-60; double maxV=30;

void setup()
{
  #ifdef ULTRASONIDO
  setupTRACKER3D();
  #endif

  #ifdef TRACKER2D
  setupTRACKER2D();
    #ifdef MOTOR2D
    setupMOTOR1();
    #endif
  #endif

  #ifdef TRACKER3D
  setupTRACKER3D();
    #ifdef MOTOR3D
```

```
    setupMOTOR1();
    setupMOTOR2();
    #endif
  #endif

  #ifdef MOTOR
  setupMOTOR1();
  setupMOTOR2();
  #endif

  #ifdef LED
  setupLED(1); //Activa LED
  #endif

  Serial.begin(9600);
}

int test = 0;
void loop(){

  #ifdef MATHS  //ALGORITHM TESTING
  double dist1= 0.5081;
  double dist2= 0.5120;

  double result1 = computeAngleV1(dist1, dist2);
  double result2 = computeAngleV2(dist1, dist2);
  double result3 = computeAngleV3(dist1, dist2);
  Serial.print(result1,2); Serial.print(" "); Serial.print(result2,2); Se
rial.print(" "); Serial.println(result3,2);
  #endif

  #ifdef MOTOR //MOTOR TESTING
  double angle = 0;
  //MOTOR 1 (HORIZONTAL) - angle > 0 : sentido antihorario
  //MOTOR 2 (VERTICAL) - angle > 0 : subo

  if (test==0){

    moverMOTOR1(-90);
    moverMOTOR1(180);
    moverMOTOR1(-90);

    moverMOTOR2(45);
    moverMOTOR2(-90);
    moverMOTOR2(45);

  }
  test=1;
   #endif
```

```
  // **** TRACKER 2D *****
  #ifdef TRACKER2D
  double dist1 = distSENSOR1(); delay(20);
  double dist2 = distSENSOR2(); delay(20);

  double angle_hor = computeAngleV3(dist1/100, dist2/100);

      #ifdef MOTOR2D
      if (abs(angle_hor)<=umbral) //No superamos el umbral horizontal
      moverMOTOR1(angle_hor);

      //CASO FUERA DE HAZ ANGULAR - ALGORITMOS NO FUNCIONAN (RESULTADO NA
N)
      if (isnan(angle_hor)==1)
     {
        if ((dist1>=dist2)&&(actualH+umbral<maxH))
        angle_hor = umbral; //Objetivo a la derecha
        else if ((dist1<dist2)&&(actualH-umbral>minH))
        angle_hor = -1*umbral; //Objetivo a la izquieda
        else
        angle_hor = 0; //Supera rango operativo

      moverMOTOR1(angle_hor);
     }
        #endif
#endif

  // **** TRACKER 3D *****
  #ifdef TRACKER3D

  double dist1 = distSENSOR1(); delay(20); //Delay para eliminar ecos (20
 ms = 6,8 m)
  double dist2 = distSENSOR2(); delay(20); //Evita que los sensores se in
terfieran
  double dist3 = distSENSOR3(); delay(20);
  double dist4 = distSENSOR4(); delay(20);

  double angle_hor = computeAngleV3(dist1/100, dist2/100);
  double angle_ver = computeAngleV3(dist3/100, dist4/100);

    #ifdef MOTOR3D
       double umbral = 10; //Umbral máximo de haz ANGULAR

//Si al motor se le aplica un ángulo > 0 -> GIRA ANTIHORARIO
//angle_hor > 0 => Objetivo a la derecha (MOTOR DEBE GIRAR ANTIHORARIO) -
> OK
      //angle_ver > 0 => Objetivo por debajo (MOTOR DEBE GIRAR HORARIO) -
> CAMBIAR SIGNO
```

```
 if (abs(angle_hor)<=umbral) //No superamos el umbral horizontal
 {
     //Comprobamos estar dentro del rango operativo
     if ((dist1>=dist2)&&(actualH+angle_hor<maxH))
     angle_hor = angle_hor;
     else if ((dist1<dist2)&&(actualH-angle_hor>minH))
     angle_hor = angle_hor;
     else
     angle_hor = 0;

     moverMOTOR1(angle_hor);
 }

 if (abs(angle_ver)<=umbral) //No superamos el umbral vertical
 {
     //Comprobamos estar dentro del rango operativo
     if ((dist3>=dist4)&&(actualV-angle_ver>minV))
     angle_ver = angle_ver;
     else if ((dist3<dist4)&&(actualV+angle_ver<maxV))
     angle_ver = angle_ver;
     else
     angle_ver = 0;

     moverMOTOR2(-1*angle_ver);
 }

//CASOS FUERA DE HAZ ANGULAR - ALGORITMOS NO FUNCIONAN (RES NAN)
if (isnan(angle_hor)==1)
{
   if ((dist1>=dist2)&&(actualH+umbral<maxH))
   angle_hor = umbral; //Objetivo a la derecha
   else if ((dist1<dist2)&&(actualH-umbral>minH))
   angle_hor = -1*umbral; //Objetivo a la izquieda
   else
   angle_hor = 0; //Supera rango operativo

 moverMOTOR1(angle_hor);
}

if (isnan(angle_ver)==1)
{
   if ((dist3>=dist4)&&(actualV-umbral>minV))
   angle_ver = -1*umbral; //Objetivo por debajo
   else if ((dist3<dist4)&&(actualV+umbral<maxV))
   angle_ver = umbral; // Objetivo por encima
   else
   angle_ver = 0; //Supera rango operativo
```

```
      moverMOTOR2(angle_ver);
     }

      actualH += angle_hor;
      actualV += angle_ver;

      delay(3);
    #endif

  #endif
}
```